GUIA DO PROGRAMADOR MSX

Dicas de programação pelo MSX-DOS

Acesso direto aos drives

Programação da MEGARAM

INCLUI 34 PROGRAMAS EXEMPLO

EDUARDO A. BARBOSA

Guia do Programador

MSX™

EDUARDO ALBERTO BARBOSA

EDITORA CIÊNCIA MODERNA LTDA.

Editor: Paulo André P. Marques
Capa e Layout: Renato Martins
Revisão: Carlos Augusto L. Almeida
Composição, Diagramação e Arte-Final: ECM

MARCAS REGISTRADAS USADAS NESTE LIVRO:

MSX e Expert são marcas registradas da Gradiente Eletrônica Ltda.
HOTBIT é marca registrada da Sharp Equipamentos Eletrônicos S.A.
DDX é marca registrada da Digital Design Eletrônica Ltda.
IBM-PC é marca registrada da International Business Machines Co.
Windows é marca registrada da Microsoft Inc.

FICHA CATALOGRÁFICA

Barbosa, Eduardo Alberto Ramalho Sampaio

Guia do Programador MSX - Eduardo Alberto Ramalho Sampaio Barbosa - Rio de Janeiro: Editora Ciência Moderna Ltda. - 1990.

1 - Guia do Programador MSX - I. TÍTULO

CDD - 001.642

1990.
EDITORA CIÊNCIA MODERNA LTDA.
Av. Rio Branco, 156 sala 713
Rio de Janeiro - RJ. 20043
Brasil.
Tel (021) 262-3111

DEDICATÓRIA

Não poderia de forma alguma deixar de dedicar este livro aos meus pais, Eduardo Machado Sampaio Barbosa e Maria Gabriela Sampaio Barbosa, como um reconhecimento pela boa educação que souberam me dar, baseada na retidão, honestidade e respeito ao próximo. Gostaria também de dedicar este livro à minha grande incentivadora, Maria Manuela Sampaio Barbosa, pelo carinho com que sempre me ajuda a vencer os maiores obstáculos.

AGRADECIMENTOS

Gostaria de agradecer ao meu editor, Paulo André P. Marques, não só pelo apoio que me deu ao longo deste projeto, como também por sua visão empresarial, que transmite um otimismo consciente a todo o grupo da Editora Ciência Moderna. Gostaria também de agradecer a Núbia Deval dos Santos e a Fernando Gilberto G. Reis pela paciente etapa de redigitação, a Carlos Augusto L. Almeida e a Rita de Cássia M.Diogo pela minuciosa revisão.

Durante a elaboração deste livro, contei com a valiosa colaboração do amigo Rafael Cansado Sanchez, um entusiasta do MSX que me transmitiu diversas sugestões para que o livro se aproximasse o máximo possível do usuário comum. A este grande amigo, gostaria de transmitir um muito obrigado especial.

Caro leitor,

Tendo em vista o elevado número de listagens de programas presente neste livro, bem como o tamanho individual de cada uma delas, a Editora Ciência Moderna lhe oferece a possibilidade de adquirí-las todas em dois disquetes de 5 1/4" (360 K) ou um disquete de 3 1/2" (720 K) - a opção de um disquete de 5 1/4" com 80 trilhas (720 K) também está disponível. Para tanto, responda ao cupom(order form) que acompanha este livro.
Além das listagens-fonte dos programas apresentados,a Editora Ciência Moderna oferece com exclusividade os seguintes programas de autoria de Eduardo Barbosa:

Top Pirate II - O melhor copiador já inventado para o MSX. **Não existe um único programa que este copiador não reproduza**. Entre as suas características, temos:

* Ambiente dirigido por menus.
* Utilização da Megaram, permitindo a cópia de um disquete de 720 K com apenas quatro trocas de disquete, se você estiver usando um só drive.
* Configuração total das operações executadas, incluindo correção de erros, tipo de cópia, drive-fonte, drive de destino, número de tentativas, etc.
* Alta velocidade: copia um disquete de 360 K em apenas 42 segundos.
* Segurança: a cópia gerada é **exatamente** igual ao disquete original.

Top Format - Trata-se de um fomatador incrementado para o MSX. Entre as suas características, temos:

* Total compatibilidade com os sistemas MSX-DOS e MS-DOS (IBM-PC).
* Rapidez : formata um disquete de 360 K em 42 segundos, e um disquete de 720 K em 1 minuto e 15 segundos.
* Verificação da integridade do disquete.
* Definição do interleave : esta opção permite que você acelere a leitura dos arquivos normais (.COM, .BAS, .BIN, etc.) em até 60 %, sem perder a compatibilidade com o MSX-DOS e com o MS-DOS.
* Reformatação do disquete sem perda de dados.
* Possibilidade de mudar o interleave do disquete sem perder dados.

Top Erase - Trata-se de um programa que literalmente torna virgem os disquetes já formatados. Ideal para ser utilizado em outros copiadores não tão inteligentes quanto o ***Top Pirate II.***

Lembre-se: A Editora Ciência Moderna é a única empresa cadastrada pelo autor para a venda dos programas acima. Se você adquirir os programas por intermédio da Ciência Moderna, receberá um certificado que lhe garantirá o direito de substituir as suas versões atuais dos programas por novas versões lançadas, pagando apenas uma Fração do custo destes lançamentos.

Preço de cada produto:

Produto		*Preço*
Listagens dos Programas-fonte		20 BTNS
Top Pirate II		25 BTNS
Top Format		20 BTNS
Top Erase		20 BTNS
Todo Conjunto	(Top Pirate II, Top Format, Top Erase e Listagens dos Programas-Fonte)	75 BTNS

SUMÁRIO

INTRODUÇÃO

O SUCESSO DA NOVIDADE

Desde a sua chegada ao Brasil, no Natal de 1985, o sistema MSX tem provado ser o mais racional sistema de 8 bits existente. Vejamos os motivos: Ao fim de quatro anos, os dois representantes desta linha no Brasil ainda mostram um fôlego invejável, principalmente se levarmos em conta que os seus concorrentes diretos já não são sequer fabricados. Qual terá sido a razão de tamanho sucesso, que provocou a retirada de computadores famosos do mercado? Acho que todos os usuários do MSX são unânimes em afirmar que a principal vantagem do sistema está na relação preço/benefício. Dentre as inúmeras vantagens, podemos destacar o som em três canais, a imagem em alta resolução com 16 cores, que não requer monitores especiais, o microprocessador Z-80 com toda a sua ampla biblioteca de softwares, o melhor BASIC entre as máquinas de 8 bits, a utilização do CP/M como sistema de DOS, o que permite utilizar a enorme quantidade de programas feitos para este sistema, a arquitetura aberta que permite o lançamento de periféricos por fabricantes autônomos, a compatibilidade a nível de arquivos ASCII e de dados com o IBM PC, etc. Considerando-se o preço relativamente baixo da máquina em comparação com as vantagens oferecidas, fica fácil perceber o porquê de tamanho sucesso.

CRESCE O APOIO AO MSX

Prevendo o enorme sucesso da linha MSX no Brasil, algumas editoras começaram a lançar no mercado livros específicos para esta linha. Inicialmente, os livros se baseavam quase que exclusivamente em traduções de trechos do manual da Microsoft para o MSX, que ainda hoje não está disponível para o usuário comum. Entretanto, a partir de 1987 a bibliografia nacional começou a adquirir uma personalidade própria com o lançamento de livros que realmente procuravam desvendar os segredos do MSX. Hoje, o usuário dispõe de um grande número de livros realmente úteis, que podem auxiliar na programação do MSX em qualquer nível.

Na parte de software, somente a partir do segundo trimestre de 1988 é que começaram a surgir os primeiros produtos nacionais de qualidade. A demora no apoio do software teve e tem dois grandes motivos : a lentidão na regulamentação da lei de software que protegeria o direito autoral e a baixa remuneração do programador. Mas, apesar de tudo, surgiram softwares de primeira linha, como o MSX-DOS Tools nacional, o Hello, o Prokit, o Graphos III e o Cartoon, entre outros. Pela primeira vez, o usuário brasileiro começou a contar com programas em que, para responder SIM a uma pergunta, tenha de apertar a tecla S, e não a tecla Y.

PARA ONDE CAMINHA O MSX?

Tanto sucesso animou os fabricantes no exterior a lançarem novas versões. Primeiramente lançaram a versão 2.0, que se diferenciava da 1.0 (o modelo existente no Brasil) por oferecer mais recursos gráficos, dos quais podemos destacar a digitalização de imagens, a maior resolução gráfica e o maior número de cores. O BASIC da versão 2.0 também foi incrementado, para manipular estas novas características. O chip de som e o microprocessador continuaram os mesmos. Tendo em vista a aprovação do mercado, os mesmos fabricantes lançaram no final de 1988 a versão 2+ (dois plus), que incrementa ainda mais as características gráficas da versão 2.0 e apresenta uma qualidade de digitalização próxima à do vídeo real. Muito bem, isto aconteceu lá fora... E aqui? É fácil prever que estas novidades cedo ou tarde chegarão até nós, mas quando? Até agora, o que temos são boatos de que os dois fabricantes nacionais, SHARP e GRADIENTE, pretendem lançar o modelo 2.0. O que nos resta é esperar ansiosamente que os boatos se confirmem e que a SHARP retorne à produção normal, como fez a GRADIENTE. Acho que podemos ter alguma esperança, pois o que acontece lá fora acaba se refletindo aqui mais tarde; foi assim com o TRS-80, com o Apple, com o Sinclair, com o PC, e não será diferente com o MSX. Enquanto isso, podemos aproveitar o lançamento de novas placas feitas por empresas paralelas. Existem duas empresas paulistas oferecendo produtos de ótima qualidade, entre eles uma placa que permite a transformação do MSX-1 em MSX-2. Como necessito estar sempre atualizado, acabei fazendo a transformação no meu MSX. O que pude notar de imediato foi a excelência do acabamento dado à placa de transformação. Ela é limpa e sem jumps, refletindo um projeto maduro e confiável. Pelo que pude constatar, a compatibilidade com o MSX 2.0 é total e o que mais me chamou a atenção foi a possibilidade de poder visualizar o modo de alta resolução usando um televisor comum. Esta característica representa um avanço, pois não requer um investimento adicional para aproveitar as novas vantagens do processador de vídeo. Outro fato que merece destaque é a implementação de um circuito de relógio real (com bateria e tudo!), que acaba se revelando extremamente útil no manuseio de arquivos (em disco, obviamente). Fico devendo um livro dedicado única e exclusivamente ao MSX 2.0.

O QUE ESTE LIVRO PRETENDE ?

Ao realizar este livro, tive em mente aquele usuário que deseja escrever os seus próprios programas com a mesma qualidade dos softwares comerciais. Trocando em miúdos, este livro se destina a todos aqueles que querem dotar os seus programas de janelas, menus, inversões de caracteres, interrupções, etc, e que até hoje não sabem como fazê-lo. Às vezes, aparecem alguns programas interessantes nas revistas que apresentam tais recursos, mas aí a frustração é grande, pois espera-se pelo menos uma listagem do programa-fonte de tais rotinas, e o que se apresenta é uma listagem enorme de códigos em hexadecimal. Ora, isto é o mesmo que pretender que alguém aprenda a dirigir sem nunca pegar num carro. Este livro apresenta tanto o código-fonte (o programa na forma de mnemônicos) como o código-objeto (os famosos códigos em hexadecimal resultantes da compilação do código-fonte). Aqui cabe uma correção: tenho visto muitos autores confundirem os termos ingleses assembly e assembler. O primeiro se refere à linguagem propriamente dita, e o segundo ao programa montador

que obterá os códigos-fonte, legíveis para o homem, e os transformará em códigos-objeto, legíveis para a máquina. O assembler tem vários representantes, como os programas SIMPLE, DEVPAC-80, M80, etc. Cabe também lembrar que o termo linguagem de máquina está associado ao código executável (.BIN, .COM). Feita a correção, gostaria que você soubesse que a maior parte do livro teve como base o grande número de pedidos de informações que os leitores dos meus dois livros anteriores me fizeram através de cartas. A todos eles o meu muito obrigado pelas sugestões. Fica, então, renovado o convite de trocarmos informações usando a minha caixa postal. Para me escrever, envie a sua carta à:

Caixa Postal: 37669
Rio de Janeiro-RJ
CEP:22642

AS NECESSIDADES DE CONHECIMENTO E DE EQUIPAMENTO

Este livro foi escrito visando três espécies distintas de usuários:

1.O usuário iniciante que quer contar de imediato com uma ampla biblioteca de rotinas prontas e testadas para implementação futura.

2.O usuário de nível médio que deseja ampliar os seus conhecimentos estudando a construção de rotinas mais complexas.

3.O usuário profissional que não quer perder tempo desenvolvendo rotinas que já existem (no caso, todas as rotinas apresentadas neste livro - existem mais de 100).

Em todo o livro, suponho que o assemblador adotado seja o DEVPAC-80, que, na minha opinião, é simplesmente o melhor pacote já desenvolvido para o MSX. Todos os programas apresentam uma listagem do código-fonte comentada (quase sempre com mais de 6 rotinas novas), uma listagem do código-objeto em linhas DATA e um programa de teste. Vale lembrar que o programa de teste não tem a pretensão de ser completo, mas sim de apenas demonstrar o uso das rotinas implementadas pelo código-fonte.

Todos os programas foram testados nos microcomputadores EXPERT e HOTBIT, além de um EXPERT transformado em 2.0 através da placa DDX. Os programas que acessam a MEGARAM foram testados nos mesmos três tipos de computador usando a MEGARAM fabricada pela empresa DDX. Cabe lembrar que as rotinas que acessam o vídeo supõem que ele esteja em 40 ou em 80 colunas; qualquer outro valor resultará em conseqüências catastróficas (a nível visual apenas). Como a maioria das listagens é extensa demais, a **EDITORA CIÊNCIA MODERNA** oferece a possibilidade de você adquirir os disquetes com as listagens de todos os programas apresentados, bastando remeter o ORDER FORM que se encontra no final deste livro.

Quanto ao número de dicas e macetes existentes neste livro, fica difícil precisar, mas eu contei 138 entre as listagens em BASIC e assembly, rotinas completas, endereços de memória,

rotinas da BIOS, rotinas do sistema de disco, etc. Em suma, a partir de agora você pode contar com uma biblioteca razoável de novas rotinas para a implementação dos seus programas em BASIC e em assembly. Fica aqui o meu desejo de que você aproveite todo este trabalho, que foi, em essência, um trabalho de pesquisa e, na sua maioria, inédito.

Capítulo 1

O VÍDEO

Francamente, acho que um bom programa tem de apresentar telas bonitas e, acima de tudo, eficientes. O estilo de programação que usa a seleção de opções através do teclado (presssionar 1,2,3... para selecionar as opções nos menus) e não através das teclas do cursor, do joystick, ou do mouse, já está ultrapassado. Hoje em dia, é quase um regra geral a utilização do vídeo reverso para destacar a escolha atual dentro de um menu de opções. Como se vê, as telas desempenham um papel cada vez mais importante dentro do projeto de um programa. Esta é a razão pela qual começo este livro apresentando rotinas para o VDP (processador de vídeo do MSX).

Hoje, existe uma grande discussão sobre qual é a melhor interface com o usuário. Ao que tudo indica, a escolha unânime recairá sobre a interface gráfica. Mas, o que vem a ser uma interface gráfica? Aqueles que conhecem o mundo do PC, Macintosh e Amiga já tiveram, provavelmente, algum contato com o ambiente Windows (sistema operacional baseado numa interface gráfica). Este sistema operacional torna muito mais fáceis as tarefas para o usuário. Em vez do usuário digitar, por exemplo, o comando DIR B: para saber o conteúdo do disco no drive B, basta selecionar o ícone (figura) correspondente à este drive através do mouse ou teclado, pressionando em seguida um botão do mouse ou a tecla Return. Este é apenas um exemplo simples da potencialidade desses sistemas. Entre outras características, poderia destacar a capacidade de multiprocessamento (execução simultânea de vários programas) e a integração entre os diversos programas que compõem o sistema. Esta integração é muito interessante por permitir a troca de dados entre, por exemplo, um aplicativo de desenhos e um processador de textos.

Após a introdução acima, você deve estar se perguntando: "Por que nunca vi nada parecido no MSX?" A principal razão é que torna-se difícil projetar programas tão complexos usando somente 64Kb. "Então, quer dizer que nunca terei este tipo de programa no MSX?" Não necessariamente. A criatividade está aí para mostrar que não existem limites. Veja, por exemplo, o caso do computador Apple II, que não possui tantos recursos como o MSX e, no entanto, apresenta programas que utilizam a interface gráfica. Uma sugestão para quebrar a barreira dos 64Kb é usar a memória de disco em sacrifício da velocidade de execução do programa.

Como vimos, a interface gráfica realmente permite uma utilização mais simples do computador como um todo (equipamento + sistema operacional), embora se trate de um sistema operacional e não de um simples programa. Isto quer dizer, que no caso do MSX, teríamos de projetar todo um novo sistema operacional que permitisse o uso de tal interface. Como, sinceramente, não acredito que alguém financie tal projeto para uma máquina como o MSX, devemos considerar o que já existe. Assim sendo, o que temos é um equipamento MSX (1 ou 2) com as seguintes características:

	MSX 1	MSX 2
Modo Texto:	40 colunas 24 linhas 2 cores	40, 80 colunas 24 linhas 2 cores
Modo Gráfico	2:256 x 192 pontos 16 cores	256 x 192 pontos 16 cores

Tabela 1.1:Características dos modos de tela do VDP

Observe que a tabela está propositalmente incompleta, tanto para o MSX 1 como para o MSX 2. Acontece que, neste capítulo, me concentrarei somente nestes dois modos de tela, por serem os mais usados nos programas aplicativos. No Apêndice A, você poderá encontrar uma tabela completa com todos os modos de tela de ambos os modelos.

Você já deve ter observado que, com a exceção dos programas para desenhos, todos os aplicativos utilizam o modo texto. A principal razão para tal utilização se deve ao fato de que este modo é muito mais simples de ser acessado. Vamos então a uma pequena introdução ao gerenciamento do modo texto no VDP.

Como sabemos, o VDP possui uma memória RAM própria de 16Kb no MSX 1 e de 128Kb no MSX 2. É nesta memória especial, chamada VRAM, que o VDP organiza o que aparece e o modo como aparece na tela do monitor. Basicamente, existem duas tabelas: a tabela de nomes, que contém um mapeamento completo do que aparece na tela, e a tabela de padrões, que contém os desenhos de todos os 256 (0-255) caracteres do MSX. Ao ligar o MSX, a rotina de inicialização coloca os endereços destas duas tabelas na chamada área das variáveis do sistema. Vale lembrar mais uma vez que esses endereços não fazem referência à RAM normal do MSX, mas sim à RAM de vídeo (VRAM). A Figura 1.1 mostra os nomes e endereços (na RAM normal) das variáveis do sistema que contêm informações sobre o modo texto do VDP. Aqui cabe mais uma observação: neste livro, usarei os nomes originais das variáveis do sistema dados pela Microsoft.

Nome da variável	Endereço	Conteúdo
LINLEN	#F3B0	Número atual de colunas
TXTNAM	#F3B3	Endereço na VRAM da Tabela de Nomes no modo texto
TXTCGP	#F3B7	Endereço na VRAM da Tabela de Padrões dos caracteres no modo texto
GRPNAM	#F3C7	Endereço na VRAM da Tabela de Nomes no modo gráfico 2
GRPCOL	#F3C9	Endereço na VRAM da Tabela de Cores no modo gráfico 2

Tabela 1.2:Variáveis do sistema utilizadas neste capítulo

Nome da variável	Endereço	Conteúdo
GRPCGP	#F3CB	Endereço na VRAM da Tabela de Padrões dos caracteres no modo gráfico 2
GRPATR	#F3CD	Endereço na VRAM da Tabela de Atributos dos sprites no modo gráfico 2
GRPPAT	#F3CF	Endereço na VRAM da Tabela de Padrões dos sprites no modo gráfico 2
CSRY	#F3DC	Posição Y+1 do cursor na tela no modo texto
CSRX	#F3DD	Posição X+1 do cursor na tela no modo texto
SCRMOD	#FCAF	Modo atual da tela (0=modo texto, 1=modo gráfico 1, 2=modo gráfico 2, etc).

Tabela 1.2:Variáveis do sistema utilizadas neste capítulo(continuação)

As informações acima representam tudo o que precisamos para começar a projetar alguns efeitos especiais que não estão "disponíveis" no MSX. Você já deve ter observado que os programas aplicativos mais recentes fazem uso de uma técnica chamada menus pull-down. O que vem a ser isto? Esta técnica apresenta uma linha com opções, que uma vez selecionadas, fazem surgir um menu embaixo da própria opção, que, por sua vez, apresenta opções adicionais. Esta técnica faz uso, portanto, do vídeo reverso (para destacar a atual opção) e das teclas do cursor, joystick ou mouse (para selecionar as opções no menu). Sendo assim, para empregá-la precisaremos desenvolver antes de tudo uma rotina que permita a simulação do vídeo reverso no MSX.

A TABELA DE FORMAÇÃO DOS CARACTERES

Como você já sabe (se tiver alguma dúvida, consulte o livro "***Introdução à Linguagem de Máquina para MSX***", de minha autoria), o desenho de um caractere é formado por uma matriz de 8x8 bits. Tendo em vista que 8 bits formam 1 byte, cada desenho de caractere ocupa 8 bytes da memória VRAM. A função do VDP é pesquisar na tabela de padrões o desenho de um determinado caractere para acender na tela os pontos (bits=1) que o formam. Tomemos como exemplo a letra A. Na Figura 1.2, você pode observar como esta letra é desenhada usando-se o mapeamento por bits. Nesta figura, os bits iguais a 1 correspondem aos pontos que serão acesos pelo VDP, ao passo que os bits iguais a 0 permanecerão apagados.

BITS 76543210	Valor em decimal
00110000	48
01001000	72
10000100	132
10000100	132
11111100	252
10000100	132
10000100	132
00000000	0

Figura 1.2: Desenho do caractere A usando o mapeamento por bits.

Considerando tudo isto, o que poderíamos fazer para criar o vídeo reverso? Leve em conta que os bits iguais a 1 no desenho acima correspondem aos pontos que serão acesos pelo VDP, a fim de colocar o caractere na tela. Acho que agora fica mais fácil, não? Basta inverter (trocar os bits iguais a 1 por zero e vice-versa) os bytes que formam o desenho do caractere para que o VDP acenda o fundo (bits iguais a 0 no desenho da Figura 1.2) e apague o desenho do caractere (bits iguais a 1 na Figura 1.2). O desenho invertido está na Figura 1.3. Esta inversão de bits pode ser realizada empregando-se a instrução **CPL** do Z-80. Agora que a teoria já está entendida, vamos passar à prática.

BITS 76543210	Valor em decimal
11001111	207
10110111	183
01111011	123
01111011	123
00000011	3
01111011	123
01111011	123
11111111	255

Figura 1.3: Desenho invertido do caractere A usando o mapeamento por bits.

Na prática, as coisas se complicam um pouco. Como o MSX só possui 256 (0-255) caracteres, temos de alterar o desenho original de alguns deles para copiar nessas posições (lembre-se que os desenhos nada mais são do que conjuntos de 8 bytes) o desenho do caractere a inverter. Por exemplo, suponha que desejemos inverter a string "Eduardo Alberto". O que eu proponho é inverter a letra E e colocar este desenho invertido nas posições ocupadas pelo desenho do caractere 224 (#E0), inverter a letra d e colocar o desenho dela invertido nas posições ocupadas pelo desenho do caractere 225 (#E1), e assim por diante. "Mas, Eduardo,

desse jeito estaremos perdendo os desenhos dos caracteres a partir de 224!" Certo, mas esse conjunto de caracteres (224 a 255) corresponde a caracteres especiais que raramente são utilizados. Desta forma,poderemos inverter strings de até 32 caracteres de comprimento, pois de 224 a 255 temos exatamente 32 caracteres. "Mas, por que todo esse trabalho?" Acontece que no MSX, ao contrário do PC, por exemplo, só possuímos 2 cores para toda a tela, não podendo, portanto, definir uma cor para a frente (bits iguais a 1) e outra para o fundo (bits iguais a 0) de cada caractere individualmente. O único recurso que nos resta é inverter literalmente o desenho dos caracteres que formam a string. Como você já deve ter percebido, ainda existe uma outra tarefa a realizar, além da inversão. Os códigos em ASCII da string invertida não serão os mesmos da string original, logo, temos de alterar também a própria string. Por exemplo, se a string possuir um único caractere, digamos A, o código ASCII dessa string será 65, embora o código ASCII do caractere invertido seja 224. Como resolver esse problema? A solução é bem simples. Para isto, vamos estudar como o MSX manipula as strings.

AS STRINGS NO MSX

Uma string no MSX está associada a um ponteiro que indica um endereço na RAM. Este endereço nada mais é do que uma área de 3 bytes com as seguintes funções:

Byte 1: Este byte contém o comprimento da própria string.

Bytes 2 e 3: Estes dois bytes contêm o endereço da própria string na RAM.

Assim sendo, o comando **VARPTR** do BASIC retorna exatamente com o endereço inicial, a partir do qual se encontram os bytes acima mencionados. Acho que já está claro que a nossa rotina deve obter o endereço inicial da própria string, examinando o conteúdo dos bytes 2 e 3 acima mencionados, para dar início ao processo de substituição dos códigos ASCII originais pelos dos caracteres invertidos. Felizmente, como a nossa rotina está projetada para funcionar através do comando **DEFUSR** do BASIC, o próprio sistema do MSX se encarregará de fornecer a ela o ponteiro para a área de 3 bytes acima mencionada. Desta forma, para invertermos a string A$="Eduardo Alberto" bastará entrar com o comando **A$=USR(A$)** para que a variável A$ passe a conter a string já invertida.

Feitos todos os esclarecimentos, vamos à listagem da rotina em assembly do Z-80.

O PROGRAMA PARA INVERSÃO DOS CARACTERES

Listagem em assembly Z-80 do código-fonte do programa para inversão de caracteres:

```
;Programa-fonte para a inversão de caracteres. Compilar com
; o programa GEN80.COM usando a seguinte sintaxe:
;
;GEN80 PROG1.BIN=PROG1.GEN
;
;onde PROG1.GEN é o nome do arquivo-texto com esta listagem
```

```
versao  equ     #002d
linlen  equ     #f3b0
valtyp  equ     #f663
argusr  equ     #f7f8
txtcgp  equ     #f3b7

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(valtyp)      ;verifica parâmetro
        cp      #03             ;é string? Se não for
        ret     nz              ;volta para o BASIC
        ld      ix,(argusr)     ;ix=ponteiro para a string
        ld      hl,(txtcgp)     ;hl=tab. de padrões
        ld      a,(versao)      ;a=versão do MSX
        or      a               ;é igual a zero (MSX 1)?
        jr      z,inicio1       ;Sim, vai para inicio1
        ld      a,(linlen)      ;o MSX 2  está em 80 colunas?
        cp      41
        jr      c,inicio1       ;Não, pula para inicio1
        add     hl,hl           ;Sim, calcula end. da tabela
                                ;de padrões do MSX 2
inicio1
        di                      ;desabilita as interrupções
        push    hl              ;salva hl
        ld      de,#e0*8        ;calcula o endereço do
        add     hl,de           ;desenho do caractere # e 0
        ex      de,hl           ;de=end. do des. do carac. # e 0
        ld      c,#e0           ;c=#e0
        ld      b,(ix+#00)      ;b=núm. de caracs. da string
        ld      l,(ix+#01)      ;hl=endereço da string
        ld      h,(ix+#02)
        push    hl
        pop     ix              ;ix=endereço da string
        pop     hl              ;recupera hl
loop1   push    bc              ;salva bc
        push    hl              ;salva hl
        push    de              ;salva de
        ld      de,#0008        ;bytes no des. de cada carac.
        ld      b,(ix+#00)      ;b=caractere a inverter
loop2   add     hl,de           ;calcula o endereço desse
```

```
        djnz    loop2           ;caractere
        pop     de              ;recupera de
        ld      b,#08           ;prepara a modificação
loop3   call    rdvram          ;lê o byte original
        cpl                     ;inverte
        ex      de,hl           ;transfere para o novo
        call    wtvram          ;destino
        ex      de,hl
        inc     hl              ;hl=hl+1
        inc     de              ;de=de+1
        djnz    loop3           ;b=b-1
        pop     hl              ;recupera hl
        pop     bc              ;recupera bc
        ld      (ix+#00),c      ;modifica a string
        inc     c               ;c=c+1
        inc     ix              ;aponta para o próximo carac.
        djnz    loop1           ;repete até o fim da string
        ei                      ;habilita as interrupções
        ret                     ;volta para o BASIC

rdvram
        ld      a,(versao)      ;obtém a versão do MSX
        or      a               ;é MSX1?
        jr      z,rdvram1       ;Sim, vai para rdvram1
        xor     a               ;Não, inicializa o VDP
        out     (#99),a         ;do MSX2
        ld      a,#8e
        out     (#99),a
rdvram1 ld      a,l             ;informa ao
        out     (#99),a         ;VDP o endereço na
        ld      a,h             ;VRAM onde será
        and     #3f             ;lido o dado
        out     (#99),a
        ex      (sp),hl         ;demora para
        ex      (sp),hl         ;sincronização
        in      a,(#98)         ;lê o dado na VRAM
        ret

wtvram
        push    af              ;salva dado a ser gravado
        ld      a,(versao)      ;obtém a versão do MSX
        or      a               ;é MSX1?
        jr      z,wtvram1       ;Sim, vai para wtvram1
        xor     a               ;Não, inicializa o VDP
        out     (#99),a         ;do MSX2
        ld      a,#8e
        out     (#99),a
```

```
wtvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                pop     af              ;recupera o dado
                out     (#98),a         ;grava o dado
                ret

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa para inversão de caracteres:

```
10 FOR A%=&HD000 TO &HD091
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG1.BIN",&HD000,&HD091
100 DATA 3A,63,F6,FE,03,C0,DD,2A,F8,F7,2A,B7,F3,3A,2D,00
110 DATA B7,28,08,3A,B0,F3,FE,29,38,01,29,F3,E5,11,00,07
120 DATA 19,EB,0E,E0,DD,46,00,DD,6E,01,DD,66,02,E5,DD,E1
130 DATA E1,C5,E5,D5,11,08,00,DD,46,00,19,10,FD,D1,06,08
140 DATA CD,59,D0,2F,EB,CD,73,D0,EB,23,13,10,F3,E1,C1,DD
150 DATA 71,00,0C,DD,23,10,DA,FB,C9,3A,2D,00,B7,28,07,AF
160 DATA D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,E3,E3
170 DATA DB,98,C9,F5,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3
180 DATA 99,7D,D3,99,7C,E6,3F,F6,40,D3,99,E3,E3,F1,D3,98
190 DATA C9,00
```

Listagem do programa de teste:

```
10 BLOAD"PROG1.BIN"
20 DEFUSR=&HD000
30 CLS
40 A$="MSX VERSÃO 2.0"
50 LOCATE(40-LEN(A$))/2,10
60 PRINTUSR(A$)
70 IF INKEY$="" THEN FOR A%=1 TO 100:NEXT A%:GOTO 50
```

COMENTÁRIOS SOBRE O PROGRAMA PARA INVERSÃO DOS CARACTERES

Neste programa, aparecerão algumas rotinas e rótulos (labels) que podem parecer estranhos a você. Os rótulos que ainda não tínhamos visto são os seguintes:

versão aponta para o endereço #002D da ROM para sabermos a versão do MSX. (0=MSX1, 1=MSX2)

valtyp aponta para um endereço na área das variáveis do sistema que contém o tipo do argumento da função **USR** do BASIC (2=inteiro, 3=string, etc.)

argusr aponta para um endereço na área das variáveis do sistema que contém o valor passado como argumento da função **USR**, se esse argumento for inteiro ou um ponteiro para a área de 3 bytes com as informações sobre a string, como vimos anteriormente.

"Mas, se o MSX 1 e 2 são compatíveis, por que precisamos saber qual a versão do MSX?" Bem, realmente, as duas versões são compatíveis a nível de BIOS (rotinas de entrada/saída), mas não a nível de hardware. Como a nossa rotina faz um acesso direto ao VDP, e tendo em vista que os comandos de inicialização dele são diferentes de uma versão para outra, torna-se necessário saber a versão do MSX, para podermos realizar a inicialização corretamente. As rotinas que acessam a memória VRAM recebem os nomes **rdvram** e **wtvram**. A rotina **rdvram** (abreviação do inglês read vram - ***lê vram***) tem como função ler o conteúdo de um endereço na memória VRAM. Na chamada a esta rotina, o par de registros HL deverá conter o endereço da memória VRAM cujo conteúdo desejamos ler. A rotina **wtvram** (abreviação do inglês write vram - ***escreve na vram***) tem como função escrever um valor num endereço na memória VRAM. Na chamada a esta rotina, o par de registros HL deverá conter o endereço da memória VRAM cujo conteúdo será modificado. Note que o teste da versão do MSX é feito no início de cada uma destas rotinas, para realizar a inicialização correta do VDP. O programa propriamente dito é simples e acho que os comentários após os comandos na listagem do código-fonte esclarecem todas as dúvidas sobre o funcionamento. Cabe aqui apenas um pequeno esclarecimento: No início do programa, fazemos um teste para saber a versão do MSX. Se esse teste indicar que o programa está sendo executado num MSX 2, fazemos outro teste para verificar se o MSX 2 está funcionando no modo de 80 colunas. Isto se torna necessário porque, neste modo, o endereço da tabela de nomes indicado pela variável **TXTNAM** é incorreto, pois a variável informa um endereço igual a #0800, quando, na verdade, o endereço real é #1000. Como #1000 é o dobro de #0800 (lembre-se que o símbolo # indica quantidades na base hexadecimal), basta colocar a instrução **ADD HL,HL** para duplicarmos o endereço que HL está apontando. Como se vê, nem o projeto MSX está livre de bugs.

Agora que você já sabe como inverter uma string de até 32 caracteres, vejamos mais algumas rotinas que podem criar efeitos interessantes na tela no modo texto.

ROTAÇÕES DA TELA NOS QUATRO SENTIDOS

Um efeito que sempre me impressionou foi a rotação de telas nos quatro sentidos (direita, esquerda, para cima e para baixo). A rotação se caracteriza pela manutenção de todas as informações na tela, ou seja, se ela se der da direita para a esquerda, o primeiro caractere de uma linha passará a ser o último, o segundo passará a ser o primeiro, e assim por diante. Não existe perda de informações, mas sim uma reordenação das posições de tais informações. Pode parecer complicado, mas, como veremos, é bem simples.

A IMPORTÂNCIA DOS BUFFERS

Antes de iniciar a apresentação das rotinas para a rotação de telas nos quatro sentidos, gostaria de apresentar um conceito geral, mas muito útil. Você tem idéia de algum artifício para acelerar as transferências de dados? Digamos que você queira acelerar a leitura dos dados do seu disquete para a memória do computador. Qual seria, no seu entender, o modo mais rápido? Ler 1 byte por vez ou ler 128 bytes ao mesmo tempo (ou mais ainda)? O bom senso diz que a última opção é a mais rápida. Mas,por quê? A resposta é simples: Ao ler 1 byte por vez, você teria de realizar 128 operações de leitura para poder ler um bloco de 128 bytes, ao passo que na leitura de 128 bytes de uma só vez você realizaria uma única leitura para o mesmo bloco. Existe tanto uma economia de instruções (o que equivale a uma economia de memória e de tempo de execução) como também uma economia no esforço despendido pelo próprio drive (movimentação da cabeça, etc.). A este tipo de acumulação de dados, resultantes de operações de entrada/saída, dá-se o nome de bufferização. Quanto maior for o buffer, mais rápida será a operação de entrada/saída, entre outros motivos, porque a velocidade de acesso aos dados na memória RAM é muito maior do que a do acesso a qualquer periférico do computador (VRAM, drives, impressoras, etc.). "Mas, então, como saber se o buffer que eu criei é suficiente ou não?" A resposta mais correta a esta pergunta está no bom senso. Você deve usar um buffer que não seja grande demais a ponto de sacrificar as exigências de memória do seu próprio programa, e nem pequeno demais a ponto de se mostrar ineficiente para o propósito que foi criado. O ideal seria reservar uma memória à parte como uma expansão do tipo **MEGARAM** para cumprir o papel de buffer, ao invés de sacrificar os 64Kb da memória RAM de acesso direto. Mais à frente, vamos ver que podemos usar a própria VRAM como um pequeno buffer para o armazenamento temporário de dados. Enfim, os limites sempre existem, mas a criatividade pode superá-los facilmente.

ALGORITMO PARA ROTAÇÃO DA TELA DE TEXTO

Dadas estas explicações, ficou claro que o uso do buffer se torna indispensável nas operações que exijam rapidez. A minha sugestão é que, para fazer a rotação de uma linha,devemos lê-la para a memória, rotá-la aí e, por fim, reescrevê-la, já rotada, na tela do monitor. Será que você já sabe a tabela que vamos usar? É a tabela de nomes ou a tabela de padrões? Para ajudar na sua resposta, leve em consideração que, anteriormente, afirmei que a tabela de nomes informa ao VDP quais os caracteres que aparecem no vídeo e como aparecem. Acho que agora a resposta já está bem clara, não? Já que vamos trabalhar com a tabela de nomes, vamos entender o seu funcionamento. A tabela de nomes nada mais é do que um mapa do

que aparece no vídeo. Como temos , 40 colunas e 24 linhas no modo texto do MSX 1, logo temos 24*40=960 posições na tela (no MSX 2, o cálculo permanece válido se você estiver no modo de 40 colunas; em 80 colunas temos exatamente o dobro de posições). Com este cálculo, já vimos que a tabela de nomes tem exatamente 960 bytes de comprimento (em 40 colunas), destinando exatamente um byte para cada caractere que aparece na tela. Para posicionar um determinado caractere na tela, você usa o comando **LOCATE** do BASIC acompanhado das coordenadas x e y, com x variando de 0 a 39 e y de 0 a 23. Para posicionar um caractere diretamente na tabela de nomes, as coisas não são tão fáceis, já que o VDP só entende endereços para a memória VRAM. Desta forma, o mapeamento na tabela de nomes é puramente seqüencial, ou seja, a posição (0,0) do BASIC corresponde ao endereço 0 da VRAM; a posição (1,0) ao endereço 1; a posição (39,0) ao endereço 39; a posição (0,1) ao endereço 40, e assim por diante. Embora não seja necessário para as rotinas de rotação da tela, se você quiser acessar a tabela de nomes usando parâmetros do tipo x,y terá de criar uma rotina para esta finalidade (observe a rotina ***posit*** no programa que cria janelas).

Dadas as explicações, vamos às listagens dos programas.

O PROGRAMA QUE IMPLEMENTA A ROTAÇÃO PARA A ESQUERDA

Listagem em assembly Z-80 do código-fonte do programa para rotar a tela para a esquerda:

```
;Programa-fonte para a rotação da tela para a esquerda.
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG2.BIN=PROG2.GEN
;
;onde PROG2.GEN é o nome do arquivo-texto com esta listagem


versao          equ     #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN


        org     #d000
```

```
inicio
        ld      a,(valtyp)          ;verifica parâmetro
        cp      #02                 ;é inteiro?
        ret     nz                  ;Não, volta para o BASIC
        ld      a,(argusr)          ;a=número de rotações
        push    af                  ;salva número de rotações
        ld      a,(versao)          ;a=versão do MSX
        or      a                   ;é igual a zero (MSX 1)?
        jr      z,inicio1           ;Sim, vai para inicio1
        ld      a,(linlen)          ;o MSX 2 está em 80 colunas?
        cp      41
        jr      c,inicio1           ;Não, pula para inicio1
        ld      a,80                ;Sim, a=80 colunas
                jr      inicio2     ;pula para inicio2 se MSX 2
inicio1 ld      a,40                ;a=40 colunas
inicio2 ld      (numcol),a          ;coloca o núm. col. em numcol
        pop     af                  ;recupera o núm. de rotações
        di                          ;desabilita as interrupções
        ld      b,a                 ;b=número de rotações
loop1   push    bc                  ;salva contador externo
        ld      hl,(txtnam)         ;hl=end. tabela de nomes
        ld      a,(numcol)          ;a=núm. de colunas
        ld      e,a                 ;de=núm. de colunas
        ld      d,#00
        ld      b,24                ;b=24 linhas na tela
loop2   push    bc                  ;salva contador intermediário
        call    setvdprd            ;prepara o vdp para leitura
        ld      a,(numcol)          ;a=núm. de colunas
        push    de                  ;salva de
        push    hl                  ;salva hl
        ld      hl,bufferlinha      ;hl aponta para o buffer
        ld      b,a                 ;b=núm. de colunas
        call    lelinha             ;lê uma linha
        ld      a,(numcol)          ;a=núm. de colunas
        ld      hl,bufferlinha      ;hl aponta para o buffer
        ld      e,a                 ;de=núm. de colunas
        ld      d,#00
        ld      a,(hl)              ;a=primeiro carac. da linha
        add     hl,de               ;hl=aponta para o fim da
                                    ;linha+1
        ld      (hl),a              ;coloca o primeiro carac. na
                                    ;última posição
        pop     hl                  ;recupera hl
        pop     de                  ;recupera de
        call    setvdpwt            ;prepara o VDP para escrita
        push    de                  ;salva de
        push    hl                  ;salva hl
```

```
        ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
        ld      a,(numcol)          ;a=núm. de colunas
        ld      b,a                 ;b=núm. de colunas
        all     esclinha            ;envia a linha já rotada
        pop     hl                  ;recupera hl
        pop     de                  ;recupera de
        add     hl,de               ;hl aponta para a próx. linha
        pop     bc                  ;recupera bc
        djnz    loop2               ;repete para as demais linhas
        pop     bc                  ;recupera bc
        djnz    loop1               ;repete até terminar todas
                                    ;as rotações
        ei                          ;habilita as interrupções
        ret                         ;retorna ao BASIC


lelinha
        in      a,(#98)             ;lê o carac. da VRAM
        ld      (hl),a              ;salva-o no buffer
        inc     hl                  ;incrementa o ponteiro
        djnz    lelinha             ;prepara a próxima leitura
        ret                         ;retorna

esclinha
        ld      a,(hl)              ;lê o carac. do buffer
        out     (#98),a             ;escreve-o na VRAM
        inc     hl                  ;incrementa o ponteiro
        djnz    esclinha            ;prepara a próxima escrita
        ret                         ;retorna


setvdprd
        ld      a,(versao)          ;obtém a versão do MSX
        or      a                   ;é MSX1?
        jr      z,rdvram1           ;Sim, vai para rdvram1
        xor     a                   ;Não, inicializa o VDP
        out     (#99),a             ;do MSX2
        ld      a,#8e
        out     (#99),a
rdvram1 ld      a,l                 ;informa ao
        out     (#99),a             ;VDP o endereço na
        ld      a,h                 ;VRAM onde será
        and     #3f                 ;lido o dado
        out     (#99),a
        ret
```

```
setvdpwt
        ld      a,(versao)          ;obtém a versão do MSX
        or      a                   ;é MSX1?
        jr      z,wtvram1           ;Sim, vai para wtvram1
        xor     a                   ;Não, inicializa o VDP
        out     (#99),a             ;do MSX2
        ld      a,#8e
        out     (#99),a
wtvram1 ld      a,l                 ;informa ao
        out     (#99),a             ;VDP o endereço na
        ld      a,h                 ;VRAM onde o
        and     #3f                 ;dado será
        or      #40                 ;gravado
        out     (#99),a
        ret


bufferlinha     defs    81

numcol          defb    #00

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa para rotar a tela para a esquerda:

```
10 FOR A%=&HD000 TO &HD0A2
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG2.BIN",&HD000,&HD0A2
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,0B
110 DATA 3A,B0,F3,FE,29,38,04,3E,50,18,02,3E,28,32,F4,D0
120 DATA F1,F3,47,C5,2A,B3,F3,3A,F4,D0,5F,16,00,06,18,C5
130 DATA CD,75,D0,3A,F4,D0,D5,E5,21,A3,D0,47,CD,67,D0,3A
140 DATA F4,D0,21,A3,D0,5F,16,00,7E,19,77,E1,D1,CD,8B,D0
150 DATA D5,E5,21,A4,D0,3A,F4,D0,47,CD,6E,D0,E1,D1,19,C1
160 DATA 10,CD,C1,10,BE,FB,C9,DB,98,77,23,10,FA,C9,7E,D3
170 DATA 98,23,10,FA,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E
180 DATA D3,99,7D,D3,99,7C,E6,3F,D3,99,C9,3A,2D,00,B7,28
190 DATA 07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6,40
200 DATA D3,99,C9
DATA D3,99,C9
```

O PROGRAMA QUE IMPLEMENTA A ROTAÇÃO PARA A DIREITA

Listagem em assembly Z-80 do código-fonte do programa para rotar a tela para a direita:

```
;Programa-fonte para a rotação da tela para a direita.
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG3.BIN=PROG3.GEN
;
;onde PROG3.GEN é o nome do arquivo-texto com esta listagem


versao          equu    #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(valtyp)      ;verifica parâmetro
        cp      #02             ;é inteiro?
        ret     nz              ;Não, volta para o BASIC
        ld      a,(argusr)      ;a=número de rotações
        push    af              ;salva número de rotações
        ld      a,(versao)      ;a=versão do MSX
        or      a               ;é igual a zero (MSX 1)?
        jr      z,inicio1       ;Sim, vai para inicio1
        ld      a,(linlen)      ;o MSX 2 está em 80 colunas?
        cp      41
        jr      c,inicio1       ;Não, pula para inicio1
        ld      a,80            ;se estiver a=80 colunas
        jr      inicio2         ;pula para inicio2 se MSX2
inicio1 ld      a,40            ;a=40 colunas
```

```
inicio2     ld      (numcol),a          ;coloca o núm. col. em numcol
            pop     af                  ;recupera o núm. de rotações
            di                          ;desabilita as interrupções
            ld      b,a                 ;b=número de rotações
loop1       push    bc                  ;salva contador externo
            ld      hl,(txtnam)         ;hl=end. tabela de nomes
            ld      a,(numcol)          ;a=núm. de colunas
            ld      e,a                 ;de=núm. de colunas
            ld      d,#00
            ld      b,24                ;b=24 linhas na tela
loop2       push    bc                  ;salva contador intermediário
            call    setvdprd            ;prepara o VDP para leitura
            ld      a,(numcol)          ;a=núm. de colunas
            push    de                  ;salva de
            push    hl                  ;salva hl
            ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
            ld      b,a                 ;b=núm. de colunas
            call    lelinha             ;lê uma linha
            ld      a,(numcol)          ;a=núm. de colunas
            ld      hl,bufferlinha      ;hl aponta para o buffer
            ld      e,a                 ;de=núm. de colunas
            ld      d,#00
            add     hl,de               ;hl aponta para o últ. carac.
            ld      a,(hl)              ;a=último carac. da linha
            ld      hl,bufferlinha      ;hl aponta para o buffer
            ld      (hl),a              ;coloca o último caractere
                                        ;na primeira posição
            pop     hl                  ;recupera hl
            pop     de                  ;recupera de
            call    setvdpwt            ;prepara o VDP para escrita
            push    de                  ;salva de
            push    hl                  ;salva hl
            ld      hl,bufferlinha      ;hl aponta para o buffer
            ld      a,(numcol)          ;a=núm. de colunas
            ld      b,a                 ;b=núm. de colunas
            call    esclinha            ;envia a linha já rotada
            pop     hl                  ;recupera hl
            pop     de                  ;recupera de
            add     hl,de               ;hl aponta para a próx. linha
            pop     bc                  ;recupera bc
            djnz    loop2               ;repete para as demais linhas
            pop     bc                  ;recupera bc
            djnz    loop1               ;repete até terminar todas
                                        ;as rotações
            ei                          ;habilita as interrupções
            ret                         ;retorna ao BASIC
```

```
lelinha
          in      a,(#98)       ;lê o carac. da VRAM
          ld      (hl),a        ;salva-o no buffer
          inc     hl            ;incrementa o ponteiro
          djnz    lelinha       ;prepara a próxima leitura
          ret                   ;retorna

esclinha
          ld      a,(hl)        ;lê o carac. do buffer
          out     (#98),a       ;escreve-o na VRAM
          inc     hl            ;incrementa o ponteiro
          djnz    esclinha      ;prepara a próxima escrita
          ret                   ;retorna

setvdprd
          ld      a,(versao)    ;obtém a versão do MSX
          or      a             ;é MSX1?
          jr      z,rdvram1     ;Sim, vai para rdvram1
          xor     a             ;Não, inicializa o VDP
          out     (#99),a       ;do MSX2
          ld      a,#8e
          out     (#99),a
rdvram1   ld      a,l           ;informa ao
          out     (#99),a       ;VDP o endereço na
          ld      a,h           ;VRAM onde será
          and     #3f           ;lido o dado
          out     (#99),a
          ret

setvdpwt
          ld      a,(versao)    ;obtém a versão do MSX
          or      a             ;é MSX1?
          jr      z,wtvram1     ;Sim, vai para wtvram1
          xor     a             ;Não, inicializa o VDP
          out     (#99),a       ;do MSX2
          ld      a,#8e
          out     (#99),a
wtvram1   ld      a,l           ;informa ao
          out     (#99),a       ;VDP o endereço na
          ld      a,h           ;VRAM onde o
          and     #3f           ;dado será
          or      #40           ;gravado
          out     (#99),a
```

```
                ret

bufferlinha     defs    81

numcol          defb    #00

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa para rotar a tela para a direita:

```
10 FOR A%=&HD000 TO &HD0A5
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG3.BIN",&HD000,&HD0A5
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,0B
110 DATA 3A,B0,F3,FE,29,38,04,3E,50,18,02,3E,28,32,F7,D0
120 DATA F1,F3,47,C5,2A,B3,F3,3A,F7,D0,5F,16,00,06,18,C5
130 DATA CD,78,D0,3A,F7,D0,D5,E5,21,A7,D0,47,CD,6A,D0,3A
140 DATA F7,D0,21,A6,D0,5F,16,00,19,7E,21,A6,D0,77,E1,D1
150 DATA CD,8E,D0,D5,E5,21,A6,D0,3A,F7,D0,47,CD,71,D0,E1
160 DATA D1,19,C1,10,CA,C1,10,BB,FB,C9,DB,98,77,23,10,FA
170 DATA C9,7E,D3,98,23,10,FA,C9,3A,2D,00,B7,28,07,AF,D3
180 DATA 99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,C9,3A,2D
190 DATA 00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6
200 DATA 3F,F6,40,D3,99,C9
```

Listagem do programa de teste:

```
100 CLS
110 DEFINTA-Z:DIMA$(2)
120 A$(1)="esquerda!"
130 A$(2)="direita!"
140 A=2:B=2
150 DEFUSR=&HD000
160 WIDTH 40
170 GOSUB200
180 GOSUB200
190 GOTO170
200 CLS:LOCATE 1,10
210 B=BXOR3
220 PRINT"Rotação para a ";A$(B)
230 A=AXOR1
```

```
240 BLOAD"prog"+MID$(STR$(A),2,1)+".bin"
250 Z=USR(40)
260 RETURN
```

O PROGRAMA QUE IMPLEMENTA A ROTAÇÃO PARA CIMA

Listagem em assembly Z-80 do código-fonte do programa para rotar a tela para cima:

```
;Programa-fonte para a rotação da tela para a cima.
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG4.BIN=PROG4.GEN
;
;onde PROG4.GEN é o nome do arquivo-texto com esta listagem

versao          equ     #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(valtyp)      ;verifica parâmetro
        cp      #02             ;é inteiro?
        ret     nz              ;Não, volta para o BASIC
        ld      a,(argusr)      ;a=número de rotações
        push    af              ;salva número de rotações
        ld      a,(versao)      ;a=versão do MSX
        or      a               ;é igual a zero (MSX 1)?
        jr      z,inicio1       ;Sim, vai para inicio1
        ld      a,(linlen)      ;o MSX 2 está em 80 colunas?
        cp      41
```

```
            jr      c,inicio1         ;Não, pula para inicio1
            ld      a,80              ;se estiver a=80 colunas
            jr      inicio2           ;pula para inicio2 se MSX 2
inicio1     ld      a,40              ;a=40 colunas
inicio2     ld      (numcol),a        ;coloca o núm. col. em numcol
            pop     af                ;recupera o núm. de rotações
            di                        ;desabilita as interrupções
            ld      b,a               ;b=número de rotações
loop1       push    bc                ;salva contador externo
            ld      hl,(txtnam)       ;hl=end. tabela de nomes
            ld      a,(numcol)        ;a=núm. de colunas
            ld      e,a               ;de=núm. de colunas
            ld      d,#00
            ld      b,a               ;b=núm. de colunas
            call setvdprd             ;prepara o VDP para leitura
            push    hl                ;salva hl
            ld      hl,primlinha      ;hl aponta para primilinha
            call    lelinha           ;lê a primeira linha
            pop     hl                ;recupera hl
            ld      b,23              ;b=24 linhas na tela
loop2       push    bc                ;salva contador intermediário
            add     hl,de             ;hl aponta para a próx. linha
            call    setvdprd          ;prepara o VDP para leitura
            ld      a,(numcol)        ;a=núm. de colunas
            push    de                ;salva de
            push    hl                ;salva hl
            ld      hl,bufferlinha    ;hl aponta para o buffer
            ld      b,a               ;b=núm. de colunas
            call    lelinha           ;lê uma linha
            pop     hl                ;recupera hl
            pop     de                ;recupera de
            xor     a                 ;zera a flag de carry
            sbc     hl,de             ;hl aponta para a linha
                                      ;anterior
            call    setvdpwt          ;prepara o VDP para escrita
            push    de                ;salva de
            push    hl                ;salva hl
            ld      hl,bufferlinha    ;hl aponta para o buffer
            ld      a,(numcol)        ;a=núm. de colunas
            ld      b,a               ;b=núm. de colunas
            call    esclinha          ;envia a linha
            pop     hl                ;recupera hl
            pop     de                ;recupera de
            add     hl,de             ;hl aponta para a próx. linha
            pop     bc                ;recupera bc
            djnz    loop2             ;repete para as demais linhas
            call    setvdpwt          ;prepara o VDP para escrita
```

```
            ld      hl,primlinha      ;hl aponta para o buffer
                                      ;primlinha
            ld      a,(numcol)        ;a=núm. de colunas
            ld      b,a               ;b=núm. de colunas
            call    esclinha          ;envia a linha
            pop     bc                ;recupera bc
            djnz    loop1             ;repete até terminar todas
                                      ;as rotações
            ei                        ;habilita as interrupções
            ret                       ;retorna ao BASIC

lelinha
            in      a,(#98)           ;lê o carac. da VRAM
            ld      (hl),a            ;salva-o no buffer
            inc     hl                ;incrementa o ponteiro
            djnz    lelinha           ;prepara a próxima leitura
            ret                       ;retorna

esclinha
            ld      a,(hl)            ;lê o carac. do buffer
            out     (#98),a           ;escreve-o na VRAM
            inc     hl                ;incrementa o ponteiro
            djnz    esclinha          ;prepara a próxima escrita
            ret                       ;retorna

setvdprd
            ld      a,(versao)        ;obtém a versão do MSX
            or      a                 ;é MSX1?
            jr      z,rdvram1         ;Sim, vai para rdvram1
            xor     a                 ;Não, inicializa o VDP
            out     (#99),a           ;do MSX2
            ld      a,#8e
            out     (#99),a
rdvram1     ld      a,l               ;informa ao
            out     (#99),a           ;VDP o endereço na
            ld      a,h               ;VRAM onde será
            and     #3f               ;lido o dado
            out     (#99),a
            ret

setvdpwt
            ld      a,(versao)        ;obtém a versão do MSX
            or      a                 ;é MSX1?
            jr      z,wtvram1         ;Sim, vai para wtvram1
```

```
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a
wtvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde o
            and     #3f             ;dado será
            or      #40             ;gravado
            out     (#99),a
            ret

bufferlinha         defs    80

primlinha           defs    80

numcol              defb    #00

fim                 equ     $
```

Listagem em linhas DATA do código-objeto do programa para rotar a tela para cima:

```
10 FOR A%=&HD000 TO &HD0B3
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG4.BIN",&HD000,&HD0B3
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,0B
110 DATA 3A,B0,F3,FE,29,38,04,3E,50,18,02,3E,28,32,54,D1
120 DATA F1,F3,47,C5,2A,B3,F3,3A,54,D1,5F,16,00,47,CD,86
130 DATA D0,E5,21,04,D1,CD,78,D0,E1,06,17,C5,19,CD,86,D0
140 DATA 3A,54,D1,D5,E5,21,B4,D0,47,CD,78,D0,E1,D1,AF,ED
150 DATA 52,CD,9C,D0,D5,E5,21,B4,D0,3A,54,D1,47,CD,7F,D0
160 DATA E1,D1,19,C1,10,D5,CD,9C,D0,21,04,D1,3A,54,D1,47
170 DATA CD,7F,D0,C1,10,AD,FB,C9,DB,98,77,23,10,FA,C9,7E
180 DATA D3,98,23,10,FA,C9,3A,2D,00,B7,28,07,AF,D3,99,3E
190 DATA 8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,C9,3A,2D,00,B7
200 DATA 28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6
210 DATA 40,D3,99,C9
```

O PROGRAMA QUE IMPLEMENTA A ROTAÇÃO PARA BAIXO

Listagem em assembly Z-80 do código-fonte do programa para rotar a tela para baixo:

```
;Programa-fonte para a rotação da tela para baixo.
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG5.BIN=PROG5.GEN
;
;onde PROG5.GEN é o nome do arquivo-texto com esta listagem


versao          equ     #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(valtyp)      ;verifica parâmetro
        cp      #02             ;é inteiro?
        ret     nz              ;Não, volta para o BASIC
        ld      a,(argusr)      ;a=número de rotações
        push    af              ;salva número de rotações
        ld      a,(versao)      ;a=versão do MSX
        or      a               ;é igual a zero (MSX 1)?
        jr      z,inicio1       ;Sim, vai para inicio1
        ld      a,(linlen)      ;o MSX 2 está em 80 colunas?
        cp      41
        jr      c,inicio1       ;Não, pula para inicio1
        ld      a,80            ;se estiver a=80 colunas
        jr      inicio2         ;pula para inicio2 se MSX 2
inicio1 ld      a,40            ;a=40 colunas
inicio2 ld      (numcol),a      ;coloca o núm. col. em numcol
```

```
         pop    af              ;recupera o núm. de rotações
         di                     ;desabilita as interrupções
         ld     b,a             ;b=número de rotações
loop1    push   bc              ;salva contador externo
         ld     hl,(txtnam)     ;hl=end. tabela de nomes
         ld     a,(numcol)      ;a=núm. de colunas
         ld     e,a             ;de=núm. de colunas
         ld     d,#00
         ld     b,23            ;b=núm. de linhas-1
calult   add    hl,de           ;calcula o endereço da
         djnz   calult          ;última linha
         ld     b,a             ;b=núm. de colunas
         call setvdprd          ;prepara o VDP para leitura
         push   hl              ;salva hl
         ld     hl,ultlinha     ;hl aponta para o buffer
                                ;ultlinha
         call   lelinha         ;lê a última linha
         pop    hl              ;recupera hl
         ld     b,23            ;b=23 linhas na tela
loop2    push   bc              ;salva contador intermediário
         xor    a               ;zera a flag de carry
         sbc    hl,de           ;hl aponta para a linha ant.
         call   setvdprd        ;prepara o VDP para leitura
         ld     a,(numcol)      ;a=núm. de colunas
         push   de              ;salva de
         push   hl              ;salva hl
         ld     hl,bufferlinha  ;hl aponta para o buffer
         ld     b,a             ;b=núm. de colunas
         call   lelinha         ;lê uma linha
         pop    hl              ;recupera hl
         pop    de              ;recupera de
         add    hl,de           ;hl aponta para a próx. linha
         call   setvdpwt        ;prepara o VDP para escrita
         push   de              ;salva de
         push   hl              ;salva hl
         ld     hl,bufferlinha  ;hl aponta para o buffer
         ld     a,(numcol)      ;a=núm. de colunas
         ld     b,a             ;b=núm. de colunas
         call   esclinha        ;envia a linha
         pop    hl              ;recupera hl
         pop    de              ;recupera de
         xor    a               ;zera a flag de carry
         sbc    hl,de           ;hl aponta para a linha ant.
         pop    bc              ;recupera bc
         djnz   loop2           ;repete para as demais linhas
         call   setvdpwt        ;prepara o VDP para escrita
         ld     hl,ultlinha     ;hl aponta para o buffer
```

```
                                        ;primlinha
            ld      a,(numcol)          ;a=núm. de colunas
            ld      b,a                 ;b=núm. de colunas
            call    esclinha            ;escreve a última linha na
                                        ;primeira linha
            pop     bc                  ;recupera bc
            djnz    loop1               ;repete até terminar todas
                                        ;as rotações
            ei                          ;habilita as interrupções
            ret                         ;retorna ao BASIC

lelinha
            in      a,(#98)             ;lê o carac. da VRAM
            ld      (hl),a              ;salva-o no buffer
            inc     hl                  ;incrementa o ponteiro
            djnz    lelinha             ;prepara a próxima leitura
            ret                         ;retorna

esclinha
            ld      a,(hl)              ;lê o carac. do buffer
            out     (#98),a             ;escreve-o na VRAM
            inc     hl                  ;incrementa o ponteiro
            djnz    esclinha            ;prepara a próxima escrita
            ret                         ;retorna

setvdprd
            ld      a,(versao)          ;obtém a versão do MSX
            or      a                   ;é MSX1?
            jr      z,rdvram1           ;Sim, vai para rdvram1
            xor     a                   ;Não, inicializa o VDP
            out     (#99),a             ;do MSX2
            ld      a,#8e
            out     (#99),a
rdvram1     ld      a,l                 ;informa ao
            out     (#99),a             ;VDP o endereço na
            ld      a,h                 ;VRAM onde será
            and     #3f                 ;lido o dado
            out     (#99),a
            ret

setvdpwt
            ld      a,(versao)          ;obtém a versão do MSX
            or      a                   ;é MSX1?
```

```
                jr      z,wtvram1           ;Sim, vai para wtvram1
                xor     a                   ;Não, inicializa o VDP
                out     (#99),a             ;do MSX2
                ld      a,#8e
                out     (#99),a
wtvram1         ld      a,l                 ;informa ao
                out     (#99),a             ;VDP o endereço na
                ld      a,h                 ;VRAM onde o
                and     #3f                 ;dado será
                or      #40                 ;gravado
                out     (#99),a
                ret

bufferlinha             defs    80

ultlinha                defs    80

numcol                  defb    #00

fim                     equ     $
```

Listagem em linhas DATA do código-objeto do programa para rotar a tela para baixo:

```
10 FOR A%=&HD000 TO &HD0BA
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG5.BIN",&HD000,&HD0BA
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,0B
110 DATA 3A,B0,F3,FE,29,38,04,3E,50,18,02,3E,28,32,5B,D1
120 DATA F1,F3,47,C5,2A,B3,F3,3A,5B,D1,5F,16,00,06,17,19
130 DATA 10,FD,47,CD,8D,D0,E5,21,0B,D1,CD,7F,D0,E1,06,17
140 DATA C5,AF,ED,52,CD,8D,D0,3A,5B,D1,D5,E5,21,BB,D0,47
150 DATA CD,7F,D0,E1,D1,19,CD,A3,D0,D5,E5,21,BB,D0,3A,5B
160 DATA D1,47,CD,86,D0,E1,D1,AF,ED,52,C1,10,D3,CD,A3,D0
170 DATA 21,0B,D1,3A,5B,D1,47,CD,86,D0,C1,10,A6,FB,C9,DB
180 DATA 98,77,23,10,FA,C9,7E,D3,98,23,10,FA,C9,3A,2D,00
190 DATA B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F
200 DATA D3,99,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99
210 DATA 7D,D3,99,7C,E6,3F,F6,40,D3,99,C9
```

Listagem do programa de teste:

```
100 CLS
110 DEFINTA-Z:DIMA$(2)
120 A$(1)="acima!"
130 A$(2)="abaixo!"
140 A=4:B=1
150 DEFUSR=&HD000
160 WIDTH 40
170 GOSUB200
180 GOSUB200
190 GOTO170
200 CLS:LOCATE 1,10
210 B=BXOR3
220 PRINT"Rotação para a ";A$(B)
230 A=AXOR1
240 BLOAD"prog"+MID$(STR$(A),2,1)+".bin"
250 Z=USR(24)
260 RETURN
```

COMENTÁRIOS SOBRE OS PROGRAMAS DE ROTAÇÃO DA TELA

O princípio do funcionamento dos programas 2 e 3 (rotações laterais da tela) é bem simples. Em ambos os programas, a primeira coisa a fazer é detectar o ambiente onde eles estão sendo executados. Não podemos nos esquecer que as rotinas de inicialização do vídeo são diferentes entre os modelos 1 e 2 do MSX. Terminados os exames do ambiente, procede-se à leitura e à rotação linha a linha de cada uma das 24 linhas da tela. Observe que, embora possamos ter 40 ou 80 colunas, dependendo do modelo do MSX, o número de linhas estará sempre limitado a 24. Nós já vimos que o buffer em RAM acelera em muito as operações de entrada/saída. Assim sendo, a rotação da tela se processa da seguinte forma:

1.Ler toda uma linha da tela para o buffer (bufferlinha) em RAM. Neste passo, temos uma transferência da VRAM para a RAM realizada pela rotina lelinha;

2.Na rotação de tela para a esquerda, obter o primeiro caractere da linha (situado em bufferlinha) e colocá-lo na última posição (bufferlinha+40 ou bufferlinha+80);

3.Apontar o ponteiro (HL) para o início da nova linha (bufferlinha+1). Observe que o termo linha,nos passos 2 e 3,se refere à cópia da verdadeira linha no buffer em RAM. Este passo se torna necessário para que o segundo caractere da linha original na tela se torne o primeiro caractere da nova linha; o terceiro caractere se torne o segundo, e assim por diante, para dar o efeito de rotação para a esquerda;

4.Escrever a linha já rotada na memória de vídeo (VRAM). Neste passo, temos uma transferência da RAM para a VRAM realizada pela rotina esclinha, e;

5. Repetir os passos de 1 a 4 para as demais linhas da tela.

Como você pode ver, as rotinas são bem simples. Se você tiver mais alguma dúvida acompanhe os comentários nas listagens do código-fonte.

Os programas 4 e 5 (rotações verticais da tela) são igualmente simples de serem compreendidos. Até aqui, tenho usado os termos programa e rotina como sinônimos, embora não o sejam. Acontece que os programas aqui apresentados são programas na medida em que são plenamente executáveis, e rotinas no sentido de que podem ser incluídos dentro de outros programas onde serão chamados como tal. O algoritmo usado nestes dois programas é o seguinte:

1.Ler a primeira linha (no caso da rotação para cima) da tela, arquivando-a num buffer especial em RAM (primlinha). Neste passo, temos uma transferência da VRAM para a RAM;

2.Ler a segunda linha da tela, arquivando-a num buffer em RAM (bufferlinha). Neste caso, temos uma transferência da VRAM para a RAM;

3.Escrever a linha arquivada no buffer (bufferlinha) na posição ocupada pela linha anterior, ou seja, escrever a segunda linha por sobre a primeira, e assim por diante. Neste caso, temos uma transferência da RAM para a VRAM;

4.Repetir os passos 2 e 3 para as demais linhas da tela, e;

5.Escrever a linha arquivada no buffer primlinha na posição ocupada pela última linha original da tela. Neste passo, temos a última transferência da RAM para a VRAM.

Os passos acima continuam válidos para a rotação para baixo da tela, só que em vez de se salvar a primeira linha, salva-se a última. Mais uma vez, aconselho você a ler os comentários nas listagens do código-fonte dos 4 últimos programas.

ARQUIVANDO AS TELAS NA VRAM

Vamos então dar uma pequena pausa nos efeitos especiais para apresentar dois programas/rotinas muito úteis. Estou certo de que já surgiu o momento em que você desejou poder arquivar a tela atual para recuperá-la num ponto futuro dentro do seu programa. Em que algoritmo você pensou para realizar tal tarefa? Vamos raciocinar juntos. Temos dois métodos cuja escolha é quase imediata: arquivar a tela na RAM não utilizada pelo BASIC (a RAM nas páginas 0 e 1, ou seja, a RAM entre os endereços #0000 e #7FFF) usando as rotinas da BIOS, ou arquivá-la num array do tipo inteiro ou do tipo string usando a RAM disponível para o BASIC. A última hipótese não é muito aceitável, tendo em vista que a memória RAM para o BASIC já é bem pequena (cerca de 24Kb num sistema com disco). A primeira hipótese é interessante, pois deixa livre toda a memória RAM usada pelo interpretador BASIC. Mas, e se você tiver programas de extensão (aqueles ativados com a instrução CALL do BASIC) carre-

gados nessa memória, como fica a situação? Você vai concordar que a transferência de telas para esta memória irá "borrar" os programas que estiverem por lá. "Sendo assim, onde podemos arquivar as telas?" Eu tenho uma sugestão: arquivar as telas na própria memória de vídeo. Nós já vimos que no modo texto só temos duas tabelas: a tabela de nomes e a tabela de padrões. A tabela de nomes gasta 960 bytes no modo de 40 colunas e 1920 bytes no modo de 80 colunas (aplicável só ao MSX 2); já a tabela de padrões gasta 2048 bytes em ambos os modelos. Também já sabemos que o MSX possui no mínimo 16Kb (128Kb no MSX 2) de memória VRAM. Fazendo as contas, temos cerca de 13Kb livres na VRAM para usar conforme desejarmos. Você também deve estar lembrado de que a tabela de nomes contém efetivamente um mapa do que aparece e de como aparece na tela. Sendo assim, o que nos interessa arquivar é apenas a tabela de nomes que possui 960 bytes de comprimento no modo de 40 colunas e 1920 bytes no modo de 80 colunas do MSX 2. Fazendo mais algumas contas, chegamos à conclusão de que podemos arquivar 12 telas no modo texto em 40 colunas e 6 no modo em 80 colunas. Na verdade, por problemas de sincronismo só podemos arquivar 4 telas no modo texto em 80 colunas do MSX 2, apesar deste modelo possuir 128Kb de VRAM. Como a maioria dos programas trabalha somente em 40 colunas por uma questão de compatibilidade, o limite de 12 telas já é bem aceitável. Examinando os comentários nas listagens-fonte, você vai observar que, além de armazenar/recuperar as telas, também salvamos/recuperamos a posição atual do cursor na tela. Vamos então às listagens dos programas.

O PROGRAMA QUE ARQUIVA TELAS NA VRAM

Listagem em assembly Z-80 do código-fonte do programa para arquivar telas na VRAM:

```
;Programa-fonte para arquivar telas na VRAM.
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG6.BIN=PROG6.GEN
;
;onde PROG6.GEN é o nome do arquivo-texto com esta listagem

versao          equ     #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7
csry            equ     #f3dc

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN
```

```
            org     #d000

inicio
            ld      a,(valtyp)      ;verifica parâmetro
            cp      #02             ;é inteiro?
            ret     nz              ;Não, volta para o BASIC
            ld      a,(argusr)      ;a=número da tela
            push    af              ;salva número da tela
            ld      a,(versao)      ;a=versão do MSX
            or      a               ;é igual a zero (MSX 1)?
            jr      z,inicio1       ;Sim, vai para inicio1
            ld      a,(linlen)      ;o MSX2 está em 80 colunas?
            cp      41              ;
            jr      c,inicio1       ;Não, vai para inicio1
            ld      a,80            ;a=80 colunas
            ld      hl,#2000        ;hl=início da VRAM livre
            ld      de,24*80+2      ;de=tamanho da tela+2
            jr      inicio2         ;vai para inicio2
inicio1     ld      a,40            ;a=40 colunas
            ld      hl,#1000        ;hl=início da VRAM livre
            ld      de,24*40+2      ;de=tamanho da tela+2
inicio2     ld      (numcols),a     ;salva o núm. de colunas
            ld      (inivram),hl    ;salva end. inicial
            ld      (tamanho),de    ;salva tam. da tela
            ld      a,(versao)      ;a=versão do MSX
            or      a               ;é igual a zero (MSX 1)?
            jr      z,inicio3       ;Sim, vai para inicio3
            ld      a,(linlen)      ;a=núm. de colunas da tela
            cp      41              ;núm. de cols. > 40 (80 cols.)
            jr      c,inicio3       ;Não, vai para inicio3
            ld      e,4             ;Sim, e=núm. máx. de telas
                                    ;em 80 colunas
            jr      inicio4         ;vai para inicio4
inicio3     ld      e,12            ;núm. máx. de telas no MSX 1
                                    ;e 2 em 40 col
inicio4     pop     af              ;recupera o núm. da tela
            cp      e               ;Se o núm. da tela recebido
            ret     nc              ;for igual ou maior do que o
                                    ;permitido, volta.

gravatela
            di                      ;desabilita as interrupções
            push    af              ;salva todos os
            push    bc              ;registros
            push    de              ;
            push    hl              ;
            ld      hl,(inivram)    ;hl=end. inicial da VRAM
```

```
         or     a               ;núm. da tela igual a 0?
         jp     z,grava1        ;Sim, vai para grava1
         ld     de,(tamanho)    ;Não, calcula o novo end.
         ld     b,a             ;inicial
loopgrv1 add    hl,de           ;
         djnz   loopgrv1        ;
grava1   ld     b,24            ;b=24 linhas
         ld     de,(txtnam)     ;de=end. da tab. de nomes
loopgrv2 push   bc              ;salva o contador
         push   hl              ;salva end. inicial
         ld     a,(numcols)     ;a=núm. de colunas
         ld     b,a             ;b=núm. de colunas
         ld     hl,bufferlinha  ;hl aponta para o buffer
         ex     de,hl           ;troca hl por de.
                                ;hl=aponta para a VRAM
         call   setvdprd        ;prepara o VDP para leitura
         ex     de,hl           ;troca hl por de.
                                ;hl=aponta para o buf.
         call   lelinha         ;lê a linha
         pop    hl              ;recupera o end. para escrita
         push   hl              ;guarda na pilha
         call   setvdpwt        ;prepara o VDP para escrita
         ld     a,(numcols)     ;a=núm. de colunas
         ld     b,a             ;b=núm. de colunas
         ld     hl,bufferlinha  ;hl=aponta para o buffer
         call   esclinha        ;escreve a linha lida
         ld     a,(numcols)     ;a=núm. de colunas
         ld     l,a             ;hl=núm. de colunas
         ld     h,#00           ;
         add    hl,de           ;hl=end. da próx. linha
         ex     de,hl           ;de=end. da próx. linha
         ld     c,a             ;bc=núm. de colunas
         ld     b,#00           ;
         pop    hl              ;recupera end. de escrita
         add    hl,bc           ;hl=end. de escrita da próx.
                                ;linha
         pop    bc              ;recupera o contador
         djnz   loopgrv2        ;repete para as demais linhas
         call   setvdpwt        ;prepara o VDP para escrita
         ld     a,(csry)        ;a=coord. y do cursor
         out    (#98),a         ;salva a coord. na VRAM
         ld     a,(csry+#01)    ;a=coord. x do cursor
         out    (#98),a         ;salva a coord. na VRAM
         pop    hl              ;recupera os registros
         pop    de              ;salvos na pilha
         pop    bc              ;
         pop    af
```

```
          ei                       ;habilita as interrupções
          ret                      ;retorna ao BASIC

lelinha
          in       a,(#98)         ;lê o carac. da VRAM
          ld       (hl),a          ;salva-o no buffer
          inc      hl              ;incrementa o ponteiro
          djnz     lelinha         ;prepara a próxima leitura
          ret                      ;retorna

esclinha
          ld       a,(hl)          ;lê o carac. do buffer
          out      (#98),a         ;escreve-o na VRAM
          inc      hl              ;incrementa o ponteiro
          djnz     esclinha        ;prepara a próxima escrita
          ret                      ;retorna

setvdprd
          ld       a,(versao)      ;obtém a versão do MSX
          or       a               ;é MSX1?
          jr       z,rdvram1       ;Sim, vai para rdvram1
          xor      a               ;Não, inicializa o VDP
          out      (#99),a         ;do MSX2
          ld       a,#8e
          out      (#99),a         ;
rdvram1   ld       a,l             ;informa ao
          out      (#99),a         ;VDP o endereço na
          ld       a,h             ;VRAM onde será
          and      #3f             ;lido o dado
          out      (#99),a         ;
          ret

setvdpwt
          ld       a,(versao)      ;obtém a versão do MSX
          or       a               ;é MSX1?
          jr       z,wtvram1       ;Sim, vai para wtvram1
          xor      a               ;Não, inicializa o VDP
          out      (#99),a         ;do MSX2
          ld       a,#8e
          out      (#99),a
wtvram1   ld       a,l             ;informa ao
          out      (#99),a         ;VDP o endereço na
          ld       a,h             ;VRAM onde o
          and      #3f             ;dado será
          or       #40             ;gravado
          out      (#99),a         ;
```

```
                ret

numcols                 defb    #00
tamanho                 defw    #00
inivram                 defw    #00
bufferlinha             defs    80

fim                     equ     $
```

Listagem em linhas DATA do código-objeto do programa para arquivar telas na VRAM:

```
10 FOR A%=&HD000 TO &HD137
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG6.BIN",&HD000,&HD137
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,11
110 DATA 3A,B0,F3,FE,29,38,0A,3E,50,21,00,20,11,82,07,18
120 DATA 08,3E,28,21,00,10,11,C2,03,32,E2,D0,22,E5,D0,ED
130 DATA 53,E3,D0,3A,2D,00,B7,28,0B,3A,B0,F3,FE,29,38,04
140 DATA 1E,04,18,02,1E,0C,F1,BB,D0,F3,F5,C5,D5,E5,2A,E5
150 DATA D0,B7,CA,5D,D0,ED,5B,E3,D0,47,19,10,FD,06,18,ED
160 DATA 5B,B3,F3,C5,E5,3A,E2,D0,47,21,E7,D0,EB,CD,B4,D0
170 DATA EB,CD,A6,D0,E1,E5,CD,CA,D0,3A,E2,D0,47,21,E7,D0
180 DATA CD,AD,D0,3A,E2,D0,6F,26,00,19,EB,4F,06,00,E1,09
190 DATA C1,10,D0,CD,CA,D0,3A,DC,F3,D3,98,3A,DD,F3,D3,98
200 DATA E1,D1,C1,F1,FB,C9,DB,98,77,23,10,FA,C9,7E,D3,98
210 DATA 23,10,FA,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3
220 DATA 99,7D,D3,99,7C,E6,3F,D3,99,C9,3A,2D,00,B7,28,07
230 DATA AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6,40,D3
240 DATA 99,C9,00,00,00,00,00,00,00,00,00,00,00,00,00,00
250 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
260 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
270 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
280 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
290 DATA 00,00,00,00,00,00,00,00
```

O PROGRAMA QUE RECUPERA TELAS ARQUIVADAS NA VRAM

Listagem em assembly Z-80 do código-fonte do programa para recuperar telas arquivadas na VRAM:

```
;Programa-fonte para recuperar telas arquivadas na VRAM.
```

```
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG7.BIN=PROG7.GEN
;
;onde PROG7.GEN é o nome do arquivo-texto com esta listagem

versao          equ     #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7
csry            equ     #f3dc

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d200

inicio
        ld      a,(valtyp)      ;verifica parâmetro
        cp      #02             ;é inteiro?
        ret     nz              ;Não, volta para o BASIC
        ld      a,(argusr)      ;a=número da tela
        push    af              ;salva número da tela
        ld      a,(versao)      ;a=versão do MSX
        or      a               ;é igual a zero (MSX 1)?
        jr      z,inicio1       ;Sim, vai para inicio1
        ld      a,(linlen)      ;o MSX2 está em 80 colunas?
        cp      41              ;
        jr      c,inicio1       ;Não, vai para inicio1
        ld      a,80            ;a=80 colunas
        ld      hl,#2000        ;hl=início da VRAM livre
                                ;em 80 col.
        ld      de,24*80+2      ;de=tamanho da tela+2
                                ;em 80 col.
        jr      inicio2         ;vai para inicio2
inicio1 ld      a,40            ;a=40 colunas
        ld      hl,#1000        ;hl=início da VRAM livre
                                ;em 40 col.
        ld      de,24*40+2      ;de=tamanho da tela+2
                                ;em 40 col.
inicio2 ld      (numcols),a     ;salva o núm. de colunas
```

```
            ld      (inivram),hl      ;salva end. inicial
            ld      (tamanho),de      ;salva tam. da tela
            ld      a,(versao)        ;a=versão do MSX
            or      a                 ;é igual a zero (MSX 1)?
            jr      z,inicio3         ;Sim, vai para inicio3
            ld      a,(linlen)        ;a=núm. de colunas da tela
            cp      41                ;núm. de cols.40 (80 cols.)?
            jr      c,inicio3         ;Nao, vai para inicio3
            ld      e,4               ;e=núm. máx. de telas
                                      ;em 80 cols.
            jr      inicio4           ;vai para inicio4
inicio3     ld      e,12              ;núm. máx. de telas no
                                      ;MSX 1 e 2: 40 col
inicio4     pop     af                ;recupera o núm. da tela
                                      ;passado pelo BASIC.
            cp      e                 ;Se o núm. da tela recebido
            ret     nc                ;for igual ou maior do que o
                                      ;MAX. permitido, volta.
recuptela
            di                        ;desabilita as interrupções
            push    af                ;salva todos os
            push    bc                ;registros
            push    de                ;
            push    hl                ;
            ld      hl,(inivram)      ;hl=end. inicial da vram
            or      a                 ;núm. da tela igual a 0?
            jp      z,recup1          ;Sim, vai para recup1
            ld      de,(tamanho)      ;Não, calcula o novo end.
            ld      b,a               ;inicial
looprcp1    add     hl,de             ;
            djnz    looprcp1          ;
recup1      ld      b,24              ;b=24 linhas
            ld      de,(txtnam)       ;de=end. da tab. de nomes
looprcp2    push    bc                ;salva o contador
            push    hl                ;salva end. inicial
            call    setvdprd          ;prepara o VDP para leitura
            ld      a,(numcols)       ;a=núm. de colunas
            ld      b,a               ;b=núm. de colunas
            ld      hl,bufferlinha    ;hl aponta para o buffer
            call    lelinha           ;lê uma linha da tela gravada
            ex      de,hl             ;troca hl por de.
                                      ;hl=tab. de nomes
            call    setvdpwt          ;prepara o VDP para escrita
            ex      de,hl             ;troca hl por de.
                                      ;de=tab. de nomes
            ld      hl,bufferlinha    ;hl aponta para o buffer
            ld      a,(numcols)       ;a=núm. de colunas
```

```
            ld      b,a             ;b=núm. de colunas
            call    esclinha        ;escreve a linha na tela
            ld      a,(numcols)     ;a=núm. de colunas
            ld      l,a             ;hl=núm. de colunas
            ld      h,#00           ;
            add     hl,de           ;hl=end. da próx. linha
            ex      de,hl           ;de=end. da próx. linha
            ld      c,a             ;bc=núm. de colunas
            ld      b,#00           ;
            pop     hl              ;recupera end. de escrita
            add     hl,bc           ;hl=end. de leitura da próx.
                                    ;linha
            pop     bc              ;recupera o contador
            djnz    looprcp2        ;repete para as demais linhas
            call    setvdprd        ;prepara o VDP para leitura
            in      a,(#98)         ;lê a coord. y
            ld      (csry),a        ;salva a coord. y do cursor
            in      a,(#98)         ;lê a coord. x
            ld      (csry+#01),a    ;salva a coord. x do cursor
            pop     hl              ;recupera os registros
            pop     de              ;salvos na pilha
            pop     bc              ;
            pop     af
            ei                      ;habilita as interrupções
            ret                     ;retorna ao BASIC

lelinha
            in      a,(#98)         ;lê o carac. da VRAM
            ld      (hl),a          ;salva-o no buffer
            inc     hl              ;incrementa o ponteiro
            djnz    lelinha         ;prepara a próxima leitura
            ret                     ;retorna

esclinha
            ld      a,(hl)          ;lê o carac. do buffer
            out     (#98),a         ;escreve-o na VRAM
            inc     hl              ;incrementa o ponteiro
            djnz    esclinha        ;prepara a próxima escrita
            ret                     ;retorna

setvdprd
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,rdvram1       ;Sim, vai para rdvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
```

```
                out     (#99),a             ;
rdvram1         ld      a,l                 ;informa ao
                out     (#99),a             ;VDP o endereço na
                ld      a,h                 ;VRAM onde será
                and     #3f                 ;lido o dado
                out     (#99),a             ;
                ret

setvdpwt
                ld      a,(versao)          ;obtém a versão do MSX
                or      a                   ;é MSX1?
                jr      z,wtvram1           ;Sim, vai para wtvram1
                xor     a                   ;Não, inicializa o VDP
                out     (#99),a             ;do MSX2
                ld      a,#8e
                out     (#99),a
wtvram1         ld      a,l                 ;informa ao
                out     (#99),a             ;VDP o endereço na
                ld      a,h                 ;VRAM onde o
                and     #3f                 ;dado será
                or      #40                 ;gravado
                out     (#99),a             ;
                ret

numcols                 defb    #00
tamanho                 defw    #00
inivram                 defw    #00
bufferlinha             defs    80

fim                     equ     $
```

Listagem em linhas DATA do código-objeto do programa para recuperar telas arquivadas na VRAM:

```
10 FOR A%=&HD200 TO &HD335
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG7.BIN",&HD200,&HD335
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,11
110 DATA 3A,B0,F3,FE,29,38,0A,3E,50,21,00,20,11,82,07,18
120 DATA 08,3E,28,21,00,10,11,C2,03,32,E0,D2,22,E3,D2,ED
130 DATA 53,E1,D2,3A,2D,00,B7,28,0B,3A,B0,F3,FE,29,38,04
140 DATA 1E,04,18,02,1E,0C,F1,BB,D0,F3,F5,C5,D5,E5,2A,E3
150 DATA D2,B7,CA,5D,D2,ED,5B,E1,D2,47,19,10,FD,06,18,ED
160 DATA 5B,B3,F3,C5,E5,CD,B2,D2,3A,E0,D2,47,21,E5,D2,CD
```

```
170 DATA A4,D2,EB,CD,C8,D2,EB,21,E5,D2,3A,E0,D2,47,CD,AB
180 DATA D2,3A,E0,D2,6F,26,00,19,EB,4F,06,00,E1,09,C1,10
190 DATA D2,CD,B2,D2,DB,98,32,DC,F3,DB,98,32,DD,F3,E1,D1
200 DATA C1,F1,FB,C9,DB,98,77,23,10,FA,C9,7E,D3,98,23,10
210 DATA FA,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D
220 DATA D3,99,7C,E6,3F,D3,99,C9,3A,2D,00,B7,28,07,AF,D3
230 DATA 99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6,40,D3,99,C9
240 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
250 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
260 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
270 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
280 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
290 DATA 00,00,00,00,00,00
```

Listagem do programa de teste:

```
100 BLOAD"PROG6.BIN"
110 BLOAD"PROG7.BIN"
120 DEFUSR0=&HD000:REM Rotina para arquivar telas
130 DEFUSR1=&HD200:REM Rotina para recuperar telas
140 A%=PEEK(&HF3B0):IF A%41 THEN MX%=3 ELSE MX%=11
150 LG%=A%
160 FOR A%=0 TO MX%
170 CLS:LOCATE 0,2:PRINT"Tela número:";A%
180 LOCATE0,10
190 FOR B%=1 TO LG%:PRINT HEX$(A%);:NEXT B%
200 Z%=USR0(A%)
210 FOR B%=1 TO 500:NEXT B%
220 NEXT A%
230 FOR A%=0 TO MX%
240 Z%=USR1(A%)
250 FOR B%=1 TO 500:NEXT B%
260 NEXT A%
```

Observe na listagem do programa de teste que a primeira tela recebe o número 0, e não 1. Acompanhe o programa e os comentários nas listagens-fonte para tirar qualquer dúvida sobre o funcionamento destas duas rotinas.

JANELAS NA TELA DE TEXTO

Para dar aquele toque profissional aos programas, só está faltando uma rotina: aquela que abre janelas. Quando adquiri o meu PC, fiquei fascinado com o nível dos programas, já que todos eles usavam a técnica de janelas e menus drop-down. Exatamente nessa época, comecei a desenvolver o MSX-DOS Tools, que ganhou o sufixo de Nacional por ter aparecido um

conjunto de programas japoneses com exatamente o mesmo título. Uma etapa importante do meu MSX-DOS Tools foi gasta no desenvolvimento das rotinas com acesso direto às portas do disk drive e de uma rotina que simulasse com perfeição o efeito das janelas. O resultado dessa rotina apareceu após um dia de trabalho: eu não queria uma rotina que apenas desenhasse uma janela. O que eu queria era que a janela fosse se formando aos poucos, dando um efeito de explosão. A rotina **zoom** cria o efeito da explosão, desenhando e apagando janelas progressivamente a partir do ponto central da janela final. A rotina **janela** é a responsável pelo desenho das janelas propriamente ditas. A rotina **posit** é a responsável pelo posicionamento do cursor nas coordenadas passadas por hl. Para acessar as rotinas **zoom** e **janela**, o registro h deverá conter a coordenada x_1; o registro l a coordenada y_1; o registro d a coordenada x_2; e o registro e a coordenada y_2. As coordenadas x_1,y_1 correspondem ao canto superior esquerdo da janela e as coordenadas x_2,y_2 ao canto inferior direito. Na rotina **posit**, o par hl deverá conter a posição x (em h) e a posição y (em l). Todas as rotinas estão muito bem comentadas, de modo que acredito não ser difícil para você entender o funcionamento.

O PROGRAMA QUE IMPLEMENTA JANELAS NA TELA DE TEXTO

Listagem em assembly Z-80 do código-fonte do programa para abrir janelas:

```
;Programa-fonte para desenhar janelas explodindo.
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG8.BIN=PROG8.GEN
;
;onde PROG8.GEN é o nome do arquivo-texto com esta listagem

versao          equ     #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7
csry            equ     #f3dc

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000
```

```
inicio
            di                          ;desabilita as interrupções
            ld      a,(versao)          ;a=versão do MSX
            or      a                   ;é igual a zero (MSX 1)?
            jr      z,inicio1           ;se for, vai para inicio1
            ld      a,(linlen)          ;o MSX2 está em 80 colunas?
            cp      41                  ;
            ld      a,40                ;a=40 colunas
            jr      c,inicio1           ;Não, vai para inicio1
            ld      a,80                ;a=80 colunas
inicio1     ld      (numcols),a         ;salva o núm. de colunas
            ld      hl,#0000            ;h=x1,l=y1
            ld      de,#0000            ;d=x2,e=y2
            call    zoom                ;chama a rot. da janela
            ei                          ;habilita as interrupções
            ret

;A rotina zoom cria o efeito de explosão da janela

zoom
            xor     a                   ;zera o acumulador
            ld      (bufcol),a          ;zera o buffer da coluna
            ld      (buflin),a          ;zera o buffer da linha
            ld      (coror1),hl         ;salva as coordenadas x1,y1
            ld      (coror2),de         ;salva as coordenadas x2,y2
            ld      a,h                 ;carrega a com h (x1)
            add     a,d                 ;soma com d (x2) e divide
            srl     a                   ;por dois para achar Xmedio
            dec     a                   ;decrementa Xmedio
            ld      h,a                 ;carrega h (x1) com Xmedio
            inc     a                   ;incrementa o ponto médio
                                        ;para que
            inc     a                   ;x2=x1+1, onde x1=Xmedio-1
            ld      d,a                 ;carrega d com x2
            ld      a,l                 ;a=y1
            add     a,e                 ;soma com e (y2) e divide
            srl     a                   ;por dois para achar Ymedio
            dec     a                   ;decrementa Ymedio
            ld      l,a                 ;carrega l (y1) com Ymedio
            inc     a                   ;incrementa o ponto médio
                                        ;para que
            inc     a                   ;y2=y1+1, onde y1=Ymedio-1
            ld      e,a                 ;carrega e com y2
            call    janela              ;desenha a primeira janela
looptest    call    coluna              ;verifica em rel. à coluna-
                                        ;limite
            call    linha               ;verifica em rel. à linha-
```

```
                                    ;limite
            call    janela          ;desenha a janela
            call    espjanela       ;retardo para tornar o efeito
                                    ;visível
            call    teste2          ;verifica se já chegou à
                                    ;janela final
            jr      nz,looptest     ;Não, continua com a explosão
            ld      hl,(coror1)     ;imprime a janela final
            ld      de,(coror2)     ;
            call    janela          ;
            ret                     ;retorna

coluna      ld      a,(coror1+#01)  ;a=coord. x1 original
            cp      h               ;já foi atingida?
            jr      nc,fimcol       ;Sim, vai para fimcol
            dec     h               ;Não, x1=x1-1
            inc     d               ;x2=x2+1
            ret                     ;retorna
fimcol  ld  a,#01                   ;indica que a coluna-limite
            ld      (bufcol),a      ;foi atingida
            ret                     ;retorna

linha   ld  a,(coror1)              ;a=coord. y1 original
            cp      l               ;já foi atingida?
            jr      nc,fimlin       ;Sim, vai para fimlin
            dec     l               ;Não, y1=y1-1
            inc     e               ;y2=y2+1
            ret                     ;retorna
fimlin  ld  a,#01                   ;indica que a linha-limite
            ld      (buflin),a      ;foi atingida
            ret                     ;retorna

espjanela   push    af              ;salva os registros
            push    hl              ;afetados
            ld      hl,#1000        ;altere este valor para mudar
espjanela1  dec     hl              ;o retardo. Decrementa hl
            ld      a,h             ;verifica se chegou
            or      l               ;ao fim
            jr      nz,espjanela1   ;Não, volta para espjanela1
            pop     hl              ;recupera os registros
            pop     af              ;
            ret                     ;e retorna

teste2      ld      a,(bufcol)      ;verifica se a coluna-
            cp      #01             ;limite foi atingida
            jr      z,contes        ;Sim, vai para contes
            ret                     ;Não, retorna
```

```
contes          ld      a,(buflin)      ;verifica se a linha-
                cp      #01             ;limite foi atingida
                ret                     ;retorna. Flag Z será o
                                        ;indicador.

;a rotina janela é a responsável pelo desenho da própria janela no vídeo

janela
                push    bc              ;salva todos os registros
                push    de              ;alterados pela rotina
                push    hl              ;
                call    posit           ;posiciona o cursor em x1,y1
                ld      a,d             ;a=x2
                sub     h               ;a=x2-x1
                dec     a               ;a=núm. de pos. da linha do
                                        ;topo
                ld      b,a             ;b=núm. de pos. da linha do
                                        ;topo
                push    bc              ;salva núm. de pos. da linha do
                                        ;topo
                ld      a,#18           ;a=carac. do canto sup. esq.
                out     (#98),a         ;escreve o caractere
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
janel1          ld      a,#17           ;a=traço horizontal
                out     (#98),a         ;escreve a linha superior
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
                djnz    janel1          ;
                ld      a,#19           ;a=carac. do canto sup. dir.
                out     (#98),a         ;escreve o caractere
                pop     bc              ;recupera núm. de pos. da linha do
                                        ;topo
                ld      l,e             ;l=y2
                call    posit           ;coloca o cursor em x1,y2
                ld      a,#1a           ;a=caractere do canto inf.
                                        ;esquerdo
                out     (#98),a         ;escreve o caractere
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
janel2          ld      a,#17           ;a=traço horizontal
                out     (#98),a         ;escreve a linha inferior
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
                djnz    janel2          ;
                ld      a,#1b           ;a=caractere do canto inf.
                                        ;direito
```

```
         out     (#98),a         ;escreve o caractere
         pop     hl              ;recupera x1,y1
         pop     de              ;recupera x2,y2
         push    de              ;salva x2,y2
         push    hl              ;salva x1,y1
         ld      a,e             ;a=y2
         sub     l               ;a=y2-y1
         dec     a               ;a=núm. de pos. verticais
         ld      b,a             ;b=núm. de pos. verticais
         ld      a,d             ;a=x2
         sub     h               ;a=x2-x1
         dec     a               ;a=núm. de pos. horizontais
         ld      c,a             ;c=núm. de pos. horizontais
         inc     l               ;y1=y1+1
janel3   call    posit           ;posiciona o cursor
         ld      a,#16           ;a=traço vertical
         out     (#98),a         ;escreve o caractere
         push    bc              ;salva núm. de pos. verticais
         ld      b,c             ;b=núm. de pos. horizontais
janel4   ld      a,#20           ;a=caractere espaço em branco
         out     (#98),a         ;escreve o carac. para limpar
         ex      (sp),hl         ;demora para sincronização
         ex      (sp),hl         ;
         djnz    janel4          ;a janela
         ld      a,#16           ;a=traço vertical
         out     (#98),a         ;escreve o caractere
         inc     l               ;y1=y1+1
         pop     bc              ;recupera núm. de pos.
                                 ;verticais
         djnz    janel3          ;repete até acabar as linhas
                                 ;verticais
         pop     hl
         pop     de
         pop     bc
         ret

posit
         push    af              ;salva todos os
         push    bc              ;registros afetados
         push    de              ;por esta rotina
         push    hl              ;
         ex      de,hl           ;troca o conteúdo de hl pelo
                                 ;de de
         ld      hl,#0000        ;zera hl
         ld      a,e             ;a=linha
         or      a               ;é igual a zero?
         jr      z,posit2        ;Sim, vai para posit2
```

```
                push    de              ;Não, salva as coordenadas
                ld      b,a             ;b=núm. da linha
                ld      a,(numcols)     ;a=núm. de colunas
                ld      e,a             ;e=núm. de colunas
                ld      d,#00           ;de=núm. de colunas
posit1          add     hl,de           ;hl=hl+núm. de colunas
                djnz    posit1
                pop     de              ;recupera as coordenadas
posit2          ld      a,d             ;a=coluna
                or      a               ;é igual a zero?
                jr      z,posit3        ;Sim, vai para posit3
                ld      e,a             ;Não, e=coluna
                ld      d,#00           ;de=coluna
                add     hl,de           ;hl aponta para o end. da
                                        ;VRAM
                                        ;correspondente a x1,y1
posit3          call    setvdpwt        ;prepara o VDP para escrita
                pop     hl              ;recupera os registros
                pop     de              ;
                pop     bc              ;
                pop     af              ;
                ret                     ;e retorna


setvdpwt
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,wtvram1       ;Sim, vai para wtvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e           ;
                out     (#99),a         ;
wtvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a         ;
                ret

numcols                 defb    #00
bufcol                  defb    #00
buflin                  defb    #00
coror1                  defw    #0000
coror2                  defw    #0000

fim                     equ     $
```

Listagem em linhas DATA do código-objeto do programa para abrir janelas:

```
10 FOR A%=&HD000 TO &HD13D
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG8.BIN",&HD000,&HD13D
100 DATA F3,3A,2D,00,B7,28,0B,3A,B0,F3,FE,29,3E,28,38,02
110 DATA 3E,50,32,36,D1,21,00,00,11,00,00,CD,20,D0,FB,C9
120 DATA AF,32,37,D1,32,38,D1,22,39,D1,ED,53,3B,D1,7C,82
130 DATA CB,3F,3D,67,3C,3C,57,7D,83,CB,3F,3D,6F,3C,3C,5F
140 DATA CD,98,D0,CD,5F,D0,CD,6E,D0,CD,98,D0,CD,7D,D0,CD
150 DATA 8A,D0,20,EF,2A,39,D1,ED,5B,3B,D1,CD,98,D0,C9,3A
160 DATA 3A,D1,BC,30,03,25,14,C9,3E,01,32,37,D1,C9,3A,39
170 DATA D1,BD,30,03,2D,1C,C9,3E,01,32,38,D1,C9,F5,E5,21
180 DATA 00,10,2B,7C,B5,20,FB,E1,F1,C9,3A,37,D1,FE,01,28
190 DATA 01,C9,3A,38,D1,FE,01,C9,C5,D5,E5,CD,F6,D0,7A,94
200 DATA 3D,47,C5,3E,18,D3,98,E3,E3,3E,17,D3,98,E3,E3,10
210 DATA F8,3E,19,D3,98,C1,6B,CD,F6,D0,3E,1A,D3,98,E3,E3
220 DATA 3E,17,D3,98,E3,E3,10,F8,3E,1B,D3,98,E1,D1,D5,E5
230 DATA 7B,95,3D,47,7A,94,3D,4F,2C,CD,F6,D0,3E,16,D3,98
240 DATA C5,41,3E,20,D3,98,E3,E3,10,F8,3E,16,D3,98,2C,C1
250 DATA 10,E7,E1,D1,C1,C9,F5,C5,D5,E5,EB,21,00,00,7B,B7
260 DATA 28,0C,D5,47,3A,36,D1,5F,16,00,19,10,FD,D1,7A,B7
270 DATA 28,04,5F,16,00,19,CD,1E,D1,E1,D1,C1,F1,C9,3A,2D
280 DATA 00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6
290 DATA 3F,F6,40,D3,99,C9,00,00,00,00,00,00,00,00
```

Listagem do programa de teste:

```
10 BLOAD"PROG8.BIN"
20 CLS
30 X1=0:Y1=0:X2=39:Y2=20
40 POKE&HD016,Y1:POKE&HD017,X1:POKE&HD019,Y2:POKE&HD01A,X2
50 DEFUSR=&HD000:Z=USR(0)
60 IFINKEY$=""THEN60
```

COMENTÁRIOS SOBRE O PROGRAMA QUE IMPLEMENTA JANELAS

Repare que na linha 40 da listagem do programa de teste informamos à rotina da janela as coordenadas da janela desejada. Como você poderá perceber através da listagem do código-

fonte, não existe nenhum teste para as coordenadas da janela, portanto, seja cauteloso e use-as como o faria no BASIC (x entre 0 e 39 em 40 colunas, entre 0 e 79 em 80 colunas e y entre 0 e 23).

NOVOS MÉTODOS PARA LIMPAR O VÍDEO

Até aqui, você já tem rotinas para inversão de vídeo, para arquivo/recuperação de telas, para criação de janelas e para alguns efeitos especiais (rotações de telas). Você já tem tudo para implementar menus nos seus programas, mas, que tal continuarmos com mais alguns efeitos especiais? O que você acha de usar as rotinas de rotação para limpar a tela, ao invés de rotá-la? Imagine o seu micro,para limpar a tela,empurrando os caracteres para a esquerda, para a direita, para cima ou, ainda, para baixo. Seria um efeito interessante, não? Vamos então à apresentação das rotinas, começando por aquela que LIMPA a tela deslocando-a para a esquerda.

O PROGRAMA PARA LIMPAR O VÍDEO DESLOCANDO A TELA PARA A ESQUERDA

Listagem em assembly Z-80 do código-fonte do programa para LIMPAR a tela deslocando-a para a esquerda:

```
;programa para limpar a tela - esquerda.
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG9.BIN=PROG9.GEN
;
;onde PROG9.GEN é o nome do arquivo-texto com esta listagem


versao          equ     #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000
```

```
inicio
          ld      a,(valtyp)        ;verifica parâmetro
          cp      #02               ;é inteiro?
          ret     nz                ;Não, volta para o BASIC
          ld      a,(argusr)        ;a=número de rotações
          push    af                ;salva número de rotações
          ld      a,(versao)        ;a=versão do MSX
          or      a                 ;é igual a zero (MSX 1)?
          jr      z,inicio1         ;Sim, vai para inicio1
          ld      a,(linlen)        ;o MSX 2 está em 80 colunas?
          cp      41
          jr      c,inicio1         ;Não, pula para inicio1
          ld      a,80              ;Sim, a=80 colunas
          jr      inicio2           ;pula para inicio2 por ser
                                    ;MSX 2
inicio1   ld      a,40              ;a=40 colunas
inicio2   ld      (numcol),a        ;coloca o núm. de colunas em
                                    ;numcol
          pop     af                ;recupera o núm. de rotações
          di                        ;desabilita as interrupções
          ld      b,a               ;b=número de rotações
loop1     push    bc                ;salva contador externo
          ld      hl,(txtnam)       ;hl=end. tabela de nomes
          ld      a,(numcol)        ;a=núm. de colunas
          ld      e,a               ;de=núm. de colunas
          ld      d,#00             ;
          ld      b,24              ;b=24 linhas na tela
loop2     push    bc                ;salva contador intermediário
          call    setvdprd          ;prepara o VDP para leitura
          ld      a,(numcol)        ;a=núm. de colunas
          push    de                ;salva de
          push    hl                ;salva hl
          ld      hl,bufferlinha    ;hl aponta para o buffer
          ld      b,a               ;b=núm. de colunas
          call    lelinha           ;lê uma linha
          ld      a,(numcol)        ;a=núm. de colunas
          ld      hl,bufferlinha    ;hl aponta para o buffer
          ld      e,a               ;de=núm. de colunas
          ld      d,#00             ;
          ld      a,#20             ;a=carac. espaço em branco
          add     hl,de             ;hl=aponta para o fim da
                                    ;linha+1
          ld      (hl),a            ;coloca um espaço na últ.
                                    ;posição
          pop     hl                ;recupera hl
          pop     de                ;recupera de
          call    setvdpwt          ;prepara o VDP para escrita
```

```
                push    de              ;salva de
                push    hl              ;salva hl
                ld      hl,bufferlinha+1 ;hl aponta para o buffer+1
                ld      a,(numcol)      ;a=núm. de colunas
                ld      b,a             ;b=núm. de colunas
                call    esclinha        ;envia a linha já rotada
                pop     hl              ;recupera hl
                pop     de              ;recupera de
                add     hl,de           ;hl aponta para a próx. linha
                pop     bc              ;recupera bc
                djnz    loop2           ;repete para as demais linhas
                pop     bc              ;recupera bc
                djnz    loop1           ;repete até terminar todas
                                        ;as rotações
                ei                      ;habilita as interrupções
                ret                     ;retorna ao BASIC

lelinha
                in      a,(#98)         ;lê o carac. da VRAM
                ld      (hl),a          ;salva-o no buffer
                inc     hl              ;incrementa o ponteiro
                djnz    lelinha         ;prepara a próxima leitura
                ret                     ;retorna

esclinha
                ld      a,(hl)          ;lê o carac. do buffer
                out     (#98),a         ;escreve-o na VRAM
                inc     hl              ;incrementa o ponteiro
                djnz    esclinha        ;prepara a próxima escrita
                ret                     ;retorna


setvdprd
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,rdvram1       ;Sim, vai para rdvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a         ;
rdvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde será
                and     #3f             ;lido o dado
                out     (#99),a         ;
                ret
```

```
setvdpwt
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,wtvram1       ;Sim, vai para wtvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a
wtvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde o
            and     #3f             ;dado será
            or      #40             ;gravado
            out     (#99),a         ;
            ret

bufferlinha         defs    81

numcol              defb    #00

fim                 equ     $
```

Listagem em linhas DATA do código-objeto do programa para LIMPAR a tela deslocando-a para a esquerda:

```
10 FOR A%=&HD000 TO &HD0F6
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG9.BIN",&HD000,&HD0F6
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,0B
110 DATA 3A,B0,F3,FE,29,38,04,3E,50,18,02,3E,28,32,F5,D0
120 DATA F1,F3,47,C5,2A,B3,F3,3A,F5,D0,5F,16,00,06,18,C5
130 DATA CD,76,D0,3A,F5,D0,D5,E5,21,A4,D0,47,CD,68,D0,3A
140 DATA F5,D0,21,A4,D0,5F,16,00,3E,20,19,77,E1,D1,CD,8C
150 DATA D0,D5,E5,21,A5,D0,3A,F5,D0,47,CD,6F,D0,E1,D1,19
160 DATA C1,10,CC,C1,10,BD,FB,C9,DB,98,77,23,10,FA,C9,7E
170 DATA D3,98,23,10,FA,C9,3A,2D,00,B7,28,07,AF,D3,99,3E
180 DATA 8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,C9,3A,2D,00,B7
190 DATA 28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6
200 DATA 40,D3,99,C9,00,00,00,00,00,00,00,00,00,00,00,00
210 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
220 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
230 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
240 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
250 DATA 00,00,00,00,00,00,00
```

O PROGRAMA PARA LIMPAR O VÍDEO DESLOCANDO A TELA PARA A DIREITA

Listagem em assembly Z-80 do código-fonte do programa para limpar a tela deslocando-a para a direita:

```
;programa para limpar a tela - direita
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG10.BIN=PROG10.GEN
;
;onde PROG10.GEN é o nome do arquivo-texto com esta
;listagem

versao          equu    #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(valtyp)      ;verifica parâmetro
        cp      #02             ;é inteiro?
        ret     nz              ;Não, volta para o BASIC
        ld      a,(argusr)      ;a=número de rotações
        push    af              ;salva número de rotações
        ld      a,(versao)      ;a=versão do MSX
        or      a               ;é igual a zero (MSX 1)?
        jr      z,inicio1       ;Sim, vai para inicio1
        ld      a,(linlen)      ;o MSX 2 está em 80 colunas?
        cp      41
        jr      c,inicio1       ;Não, pula para inicio1
        ld      a,80            ;Sim, a=80 colunas
        jr      inicio2         ;pula para inicio2 por ser
                                ;MSX 2
inicio1 ld      a,40            ;a=40 colunas
```

```
inicio2    ld      (numcol),a          ;coloca o núm. de colunas em
                                       ;numcol
           pop     af                  ;recupera o núm. de rotações
           di                          ;desabilita as interrupções
           ld      b,a                 ;b=número de rotações
loop1      push    bc                  ;salva contador externo
           ld      hl,(txtnam)         ;hl=end. tabela de nomes
           ld      a,(numcol)          ;a=núm. de colunas
           ld      e,a                 ;de=núm. de colunas
           ld      d,#00               ;
           ld      b,24                ;b=24 linhas na tela
loop2      push    bc                  ;salva contador intermediário
           call    setvdprd            ;prepara o VDP para leitura
           ld      a,(numcol)          ;a=núm. de colunas
           push    de                  ;salva de
           push    hl                  ;salva hl
           ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
           ld      b,a                 ;b=núm. de colunas
           call    lelinha             ;lê uma linha
           ld      a,(numcol)          ;a=núm. de colunas
           ld      hl,bufferlinha      ;hl aponta para o buffer
           ld      e,a                 ;de=núm. de colunas
           ld      d,#00               ;
           add     hl,de               ;hl aponta para o últ. carac.
           ld      a,#20               ;a=carac. espaço em branco
           ld      hl,bufferlinha      ;hl aponta para o buffer
           ld      (hl),a              ;coloca o espaço na primeira
                                       ;posição
           pop     hl                  ;recupera hl
           pop     de                  ;recupera de
           call    setvdpwt            ;prepara o VDP para escrita
           push    de                  ;salva de
           push    hl                  ;salva hl
           ld      hl,bufferlinha      ;hl aponta para o buffer
           ld      a,(numcol)          ;a=núm. de colunas
           ld      b,a                 ;b=núm. de colunas
           call    esclinha            ;envia a linha já rotada
           pop     hl                  ;recupera hl
           pop     de                  ;recupera de
           add     hl,de               ;hl aponta para a próx. linha
           pop     bc                  ;recupera bc
           djnz    loop2               ;repete para as demais linhas
           pop     bc                  ;recupera bc
           djnz    loop1               ;repete até terminar todas
                                       ;as rotações
           ei                          ;habilita as interrupções
           ret                         ;retorna ao BASIC
```

```
lelinha
            in      a,(#98)         ;lê o carac. da VRAM
            ld      (hl),a          ;salva-o no buffer
            inc     hl              ;incrementa o ponteiro
            djnz    lelinha         ;prepara a próxima leitura
            ret                     ;retorna

esclinha
            ld      a,(hl)          ;lê o carac. do buffer
            out     (#98),a         ;escreve-o na VRAM
            inc     hl              ;incrementa o ponteiro
            djnz    esclinha        ;prepara a próxima escrita
            ret                     ;retorna


setvdprd
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,rdvram1       ;Sim, vai para rdvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a         ;
rdvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde será
            and     #3f             ;lido o dado
            out     (#99),a         ;
            ret

setvdpwt
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,wtvram1       ;Sim, vai para wtvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a
wtvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde o
            and     #3f             ;dado será
            or      #40             ;gravado
            out     (#99),a         ;
            ret
```

```
bufferlinha          defs    81
numcol               defb    #00

fim                  equ     $
```

Listagem em linhas DATA do código-objeto do programa para LIMPAR a tela deslocando-a para a direita:

```
10 FOR A%=&HD000 TO &HD0F9
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG10.BIN",&HD000,&HD0F9
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,0B
110 DATA 3A,B0,F3,FE,29,38,04,3E,50,18,02,3E,28,32,F8,D0
120 DATA F1,F3,47,C5,2A,B3,F3,3A,F8,D0,5F,16,00,06,18,C5
130 DATA CD,79,D0,3A,F8,D0,D5,E5,21,A8,D0,47,CD,6B,D0,3A
140 DATA F8,D0,21,A7,D0,5F,16,00,19,3E,20,21,A7,D0,77,E1
150 DATA D1,CD,8F,D0,D5,E5,21,A7,D0,3A,F8,D0,47,CD,72,D0
160 DATA E1,D1,19,C1,10,C9,C1,10,BA,FB,C9,DB,98,77,23,10
170 DATA FA,C9,7E,D3,98,23,10,FA,C9,3A,2D,00,B7,28,07,AF
180 DATA D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,C9,3A
190 DATA 2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C
200 DATA E6,3F,F6,40,D3,99,C9,00,00,00,00,00,00,00,00,00
210 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
220 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
230 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
240 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
250 DATA 00,00,00,00,00,00,00,00,00,20
```

O PROGRAMA PARA LIMPAR
O VÍDEO DESLOCANDO A TELA PARA CIMA

Listagem em assembly Z-80 do código-fonte do programa para limpar a tela deslocando-a para cima:

```
;programa para limpar a tela - para cima
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG11.BIN=PROG11.GEN
;
;onde PROG11.GEN é o nome do arquivo-texto com esta
```

```
;listagem

versao                  equu    #002d
linlen                  equ     #f3b0
txtnam                  equ     #f3b3
valtyp                  equ     #f663
argusr                  equ     #f7f8
txtcgp                  equ     #f3b7

            defb    #fe                 ;simula em CP/M
            defw    inicio              ;o cabecalho de
            defw    fim                 ;um arquivo
            defw    inicio              ;.BIN

            org     #d000

inicio
            ld      a,(valtyp)          ;verifica parâmetro
            cp      #02                 ;é inteiro?
            ret     nz                  ;Não, volta para o BASIC
            ld      a,(argusr)          ;a=número de rotações
            push    af                  ;salva número de rotações
            ld      a,(versao)          ;a=versão do MSX
            or      a                   ;é igual a zero (MSX 1)?
            jr      z,inicio1           ;Sim, vai para inicio1
            ld      a,(linlen)          ;o MSX 2 está em 80 colunas?
            cp      41
            jr      c,inicio1           ;Não, pula para inicio1
            ld      a,80                ;Sim, a=80 colunas
            jr      inicio2             ;pula para inicio2 por ser
                                        ;MSX 2
inicio1     ld      a,40                ;a=40 colunas
inicio2     ld      (numcol),a          ;coloca o núm. de colunas em
                                        ;numcol
            pop     af                  ;recupera o núm. de rotações
            di                          ;desabilita as interrupções
            ld      hl,primlinha        ;hl aponta para primlinha
            ld      de,primlinha+1      ;de=hl+1
            ld      bc,79               ;comp. de primlinha -1
            ld      (hl),#20            ;preenche primlinha com
            ldir                        ;espaços
            ld      b,a                 ;b=número de rotações
loop1       push    bc                  ;salva contador externo
            ld      hl,(txtnam)         ;hl=end. tabela de nomes
            ld      a,(numcol)          ;a=núm. de colunas
            ld      e,a                 ;de=núm. de colunas
```

```
        ld      d,#00           ;
        ld      b,23            ;b=24 linhas na tela
loop2   push    bc              ;salva contador intermediário
        add     hl,de           ;hl aponta para a próx. linha
        call    setvdprd        ;prepara o VDP para leitura
        ld      a,(numcol)      ;a=núm. de colunas
        push    de              ;salva de
        push    hl              ;salva hl
        ld      hl,bufferlinha  ;hl aponta para o buffer
        ld      b,a             ;b=núm. de colunas
        call    lelinha         ;lê uma linha
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        xor     a               ;zera a flag de carry
        sbc     hl,de           ;hl aponta para a linha
                                ;anterior
        call    setvdpwt        ;prepara o VDP para escrita
        push    de              ;salva de
        push    hl              ;salva hl
        ld      hl,bufferlinha  ;hl aponta para o buffer
        ld      a,(numcol)      ;a=núm. de colunas
        ld      b,a             ;b=núm. de colunas
        call    esclinha        ;envia a linha
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        add     hl,de           ;hl aponta para a próx. linha
        pop     bc              ;recupera bc
        djnz    loop2           ;repete para as demais linhas
        call    setvdpwt        ;prepara o VDP para escrita
        ld      hl,primlinha    ;hl aponta para o buffer
                                ;primlinha
        ld      a,(numcol)      ;a=núm. de colunas
        ld      b,a             ;b=núm. de colunas
        call    esclinha        ;escreve uma linha de espaços
        pop     bc              ;recupera bc
        djnz    loop1           ;repete até terminar todas
                                ;as rotações
        ei                      ;habilita as interrupções
        ret                     ;retorna ao BASIC


lelinha
        in      a,(#98)         ;lê o carac. da VRAM
        ld      (hl),a          ;salva-o no buffer
        inc     hl              ;incrementa o ponteiro
        djnz    lelinha         ;prepara a próxima leitura
        ret                     ;retorna
```

```
esclinha
            ld      a,(hl)          ;lê o carac. do buffer
            out     (#98),a         ;escreve-o na VRAM
            inc     hl              ;incrementa o ponteiro
            djnz    esclinha        ;prepara a próxima escrita
            ret                     ;retorna

setvdprd
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,rdvram1       ;Sim, vai para rdvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a         ;
rdvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde será
            and     #3f             ;lido o dado
            out     (#99),a         ;
            ret

setvdpwt
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,wtvram1       ;Sim, vai para wtvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a
wtvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde o
            and     #3f             ;dado será
            or      #40             ;gravado
            out     (#99),a         ;
            ret


bufferlinha         defs    80
primlinha           defs    80
numcol              defb    #00

fim                 equ     $
```

Listagem em linhas DATA do código-objeto do programa para LIMPAR a tela deslocando-a para cima:

```
10 FOR A%=&HD000 TO &HD156
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG11.BIN",&HD000,&HD156
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,0B
110 DATA 3A,B0,F3,FE,29,38,04,3E,50,18,02,3E,28,32,55,D1
120 DATA F1,F3,21,05,D1,11,06,D1,01,4F,00,36,20,ED,B0,47
130 DATA C5,2A,B3,F3,3A,55,D1,5F,16,00,06,17,C5,19,CD,87
140 DATA D0,3A,55,D1,D5,E5,21,B5,D0,47,CD,79,D0,E1,D1,AF
150 DATA ED,52,CD,9D,D0,D5,E5,21,B5,D0,3A,55,D1,47,CD,80
160 DATA D0,E1,D1,19,C1,10,D5,CD,9D,D0,21,05,D1,3A,55,D1
170 DATA 47,CD,80,D0,C1,10,B9,FB,C9,DB,98,77,23,10,FA,C9
180 DATA 7E,D3,98,23,10,FA,C9,3A,2D,00,B7,28,07,AF,D3,99
190 DATA 3E,8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,C9,3A,2D,00
200 DATA B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F
210 DATA F6,40,D3,99,C9,00,00,00,00,00,00,00,00,00,00,00
220 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
230 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
240 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
250 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
260 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
270 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
280 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
290 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
300 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
310 DATA 00,00,00,00,00,00,00
```

O PROGRAMA PARA LIMPAR O VÍDEO DESLOCANDO A TELA PARA BAIXO

Listagem em assembly Z-80 do código-fonte do programa para limpar a tela deslocando-a para baixo:

```
;programa para limpar a tela - baixo
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG12.BIN=PROG12.GEN
;
;onde PROG12.GEN é o nome do arquivo-texto com esta
```

```
;listagem

versao          equ     #002d
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(valtyp)      ;verifica parâmetro
        cp      #02             ;é inteiro?
        ret     nz              ;Não, volta para o BASIC
        ld      a,(argusr)      ;a=número de rotações
        push    af              ;salva número de rotações
        ld      a,(versao)      ;a=versão do MSX
        or      a               ;é igual a zero (MSX 1)?
        jr      z,inicio1       ;Sim, vai para inicio1
        ld      a,(linlen)      ;o MSX 2 está em 80 colunas?
        cp      41
        jr      c,inicio1       ;Não, pula para inicio1
        ld      a,80            ;Sim, a=80 colunas
        jr      inicio2         ;pula para inicio2 por ser
                                ;MSX 2
inicio1 ld      a,40            ;a=40 colunas
inicio2 ld      (numcol),a      ;coloca o núm. de colunas em
                                ;numcol
        pop     af              ;recupera o núm. de rotações
        di                      ;desabilita as interrupções
        ld      b,a             ;b=número de rotações
loop1   push    bc              ;salva contador externo
        ld      hl,(txtnam)     ;hl=end. tabela de nomes
        ld      a,(numcol)      ;a=núm. de colunas
        ld      e,a             ;de=núm. de colunas
        ld      d,#00           ;
        ld      b,23            ;b=núm. de linhas-1
calult  add     hl,de           ;calcula o endereço da
        djnz    calult          ;última linha
        push    de              ;salva de
```

```
        push    hl              ;salva hl
        ld      hl,ultlinha     ;hl aponta para ultlinha
        ld      de,ultlinha+1   ;de=hl+1
        ld      bc,79           ;bc=comp. da linha-1
        ld      (hl),#20        ;preenche com espaços
        ldir                    ;
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        ld      b,23            ;b=23 linhas na tela
loop2   push    bc              ;salva contador intermediário
        xor     a               ;zera a flag de carry
        sbc     hl,de           ;hl aponta para a linha ant.
        call    setvdprd        ;prepara o VDP para leitura
        ld      a,(numcol)      ;a=núm. de colunas
        push    de              ;salva de
        push    hl              ;salva hl
        ld      hl,bufferlinha  ;hl aponta para o buffer
        ld      b,a             ;b=núm. de colunas
        call    lelinha         ;lê uma linha
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        add     hl,de           ;hl aponta para a próx. linha
        call    setvdpwt        ;prepara o VDP para escrita
        push    de              ;salva de
        push    hl              ;salva hl
        ld      hl,bufferlinha  ;hl aponta para o buffer
        ld      a,(numcol)      ;a=núm. de colunas
        ld      b,a             ;b=núm. de colunas
        call    esclinha        ;envia a linha
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        xor     a               ;zera a flag de carry
        sbc     hl,de           ;hl aponta para a linha ant.
        pop     bc              ;recupera bc
        djnz    loop2           ;repete para as demais linhas
        call    setvdpwt        ;prepara o VDP para escrita
        ld      hl,ultlinha     ;hl aponta para o buffer
                                ;primlinha
        ld      a,(numcol)      ;a=núm. de colunas
        ld      b,a             ;b=núm. de colunas
        call    esclinha        ;escreve uma linha de espaços
        pop     bc              ;recupera bc
        djnz    loop1           ;repete até terminar todas
                                ;as rotações
        ei                      ;habilita as interrupções
        ret                     ;retorna ao BASIC
```

```
lelinha
            in      a,(#98)         ;lê o carac. da VRAM
            ld      (hl),a          ;salva-o no buffer
            inc     hl              ;incrementa o ponteiro
            djnz    lelinha         ;prepara a próxima leitura
            ret                     ;retorna
esclinha
            ld      a,(hl)          ;lê o carac. do buffer
            out     (#98),a         ;escreve-o na VRAM
            inc     hl              ;incrementa o ponteiro
            djnz    esclinha        ;prepara a próxima escrita
            ret                     ;retorna
setvdprd
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,rdvram1       ;Sim, vai para rdvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a         ;
rdvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde será
            and     #3f             ;lido o dado
            out     (#99),a         ;
            ret
setvdpwt
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,wtvram1       ;Sim, vai para wtvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a
wtvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde o
            and     #3f             ;dado será
            or      #40             ;gravado
            out     (#99),a         ;
            ret
```

```
bufferlinha          defs    80
ultlinha             defs    80
numcol               defb    #00

fim                  equ     $
```

Listagem em linhas DATA do código-objeto do programa para limpar a tela deslocando-a para baixo:

```
10 FOR A%=&HD000 TO &HD161
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG12.BIN",&HD000,&HD161
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,F5,3A,2D,00,B7,28,0B
110 DATA 3A,B0,F3,FE,29,38,04,3E,50,18,02,3E,28,32,60,D1
120 DATA F1,F3,47,C5,2A,B3,F3,3A,60,D1,5F,16,00,06,17,19
130 DATA 10,FD,D5,E5,21,10,D1,11,11,D1,01,4F,00,36,20,ED
140 DATA B0,E1,D1,06,17,C5,AF,ED,52,CD,92,D0,3A,60,D1,D5
150 DATA E5,21,C0,D0,47,CD,84,D0,E1,D1,19,CD,A8,D0,D5,E5
160 DATA 21,C0,D0,3A,60,D1,47,CD,8B,D0,E1,D1,AF,ED,52,C1
170 DATA 10,D3,CD,A8,D0,21,10,D1,3A,60,D1,47,CD,8B,D0,C1
180 DATA 10,A1,FB,C9,DB,98,77,23,10,FA,C9,7E,D3,98,23,10
190 DATA FA,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D
200 DATA D3,99,7C,E6,3F,D3,99,C9,3A,2D,00,B7,28,07,AF,D3
210 DATA 99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6,40,D3,99,C9
220 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
230 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
240 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
250 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
260 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
270 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
280 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
290 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
300 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
310 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
320 DATA 00,00
```

COMENTÁRIOS SOBRE OS PROGRAMAS PARA LIMPAR O VÍDEO

Até agora, vimos quatro programas que limpam a tela deslocando-a numa determinada direção. Agora, gostaria de apresentar uma rotina de um efeito que sempre me impressionou: limpar a tela como se estivéssemos fechando uma persiana. A rotina que apresento neste

capítulo tem um funcionamento bem simples, que consiste na rotação para cima dos desenhos de cada um dos 256 caracteres.

LIMPANDO A TELA COM EFEITO PERSIANA

Na verdade, o termo rotação está mal empregado, já que existe uma rotação para cima seguida de um preenchimento com zeros de todos os 8 bits que formam o último byte do desenho. "Mas, por que com zeros?" Você deve estar lembrado, quando vimos a tabela de padrões, que o VDP "acende" os bits iguais a 1 dos 8 bytes que formam o desenho de um caractere. Dito isto, o que nós temos que fazer é ir preenchendo paulatinamente com zeros os bits que constituem os bytes que formam os desenhos. Vamos pegar o exemplo da letra A. A rotina obteria o desenho da letra A (mostrado no início deste capítulo) e, no final da primeira iteração, modificaria o desenho para:

1a. Iteração:

01001000
10000100
10000100
11111100
10000100
10000100
00000000
00000000

Na segunda iteração, o desenho já seria:

2a. Iteração:

10000100
10000100
11111100
10000100
10000100
00000000
00000000
00000000

E assim continua até que, no final da oitava e última iteração, o desenho seria:

8a. Iteração:

00000000
00000000
00000000
00000000

00000000
00000000
00000000
00000000

"Mas, Eduardo, nós estamos modificando os desenhos dos caracteres, e não a tabela de nomes, que é o que efetivamente mapeia o que aparece no vídeo!" Você está absolutamente certo. Por isso, no final desta rotina, temos de LIMPAR efetivamente a tela, preenchendo a tabela de nomes com espaços e restaurar os desenhos originais de todos os caracteres. Estas duas últimas etapas são executadas com chamadas às rotinas CLS e INITXT, respectivamente, na ROM BIOS do seu MSX. Apresentada a teoria, vamos à listagem da rotina.

O PROGRAMA QUE LIMPA O VÍDEO COM EFEITO PERSIANA

Listagem em assembly Z-80 do código-fonte do programa para limpar a tela com efeito persiana:

```
;programa para limpar a tela - persiana
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG13.BIN=PROG13.GEN
;
;onde PROG13.GEN é o nome do arquivo-texto com esta
;listagem

versao          equ     #002d
cls             equ     #00c3
initxt          equ     #006c
linlen          equ     #f3b0
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000
```

```
inicio
            di                          ;desabilita as interrupções
            ld      hl,(txtcgp)         ;hl=end. tab. de formação dos
                                        ;caracteres
            ld      a,(versao)          ;a=versão do MSX
            or      a                   ;é igual a zero (MSX 1)?
            jr      z,inicio1           ;Sim, vai para inicio1
            ld      a,(linlen)          ;o MSX 2 está em 80 colunas?
            cp      41
            jr      c,inicio1           ;Não, pula para inicio1
            add     hl,hl               ;Sim, calcula novo end. na
                                        ;VRAM
inicio1     ld      (endtab),hl         ;salva o end. da tabela
            ld      b,#08               ;b=núm. de bytes por carac.
loop0       push    bc                  ;salva núm.de bytes por carac.
            ld      b,#00               ;b=256 repetições=núm. de
                                        ;caracteres
            ld      hl,(endtab)         ;obtém o end. da tabela em hl
            ld      de,#0008            ;de=bytes por desenho de
                                        carac.
loop1       push    bc                  ;salva bc
            push    de                  ;salva de
            push    hl                  ;salva hl
            call    setvdprd            ;prepara o VDP para leitura
            ld      b,#08               ;b=núm. de bytes por desenho
                                        ;de caractere
            ld      hl,buffer           ;hl aponta para o buffer
            call    lelinha             ;lê um desenho de carac.
            ld      (hl),#00            ;coloca um espaço no buffer
            pop     hl                  ;recupera valor antigo de hl
            push    hl                  ;torna a salvar
            call    setvdpwt            ;prepara o VDP para escrita
            ld      b,#08               ;b=núm. de bytes por desenho
                                        ;de caractere
            ld      hl,buffer+#01       ;hl aponta para buffer+#01
            call    esclinha            ;escreve o novo desenho
            pop     hl                  ;recupera os
            pop     de                  ;registros
            pop     bc                  ;salvos anteriormente
            add     hl,de               ;hl aponta para o próximo
                                        ;desenho
            djnz    loop1               ;repete para todos os 256
                                        ;caracteres
            call    espera              ;retardo para tornar o efeito
                                        ;visível
            pop     bc                  ;recupera o contador externo
            djnz    loop0               ;repete enquanto b < > 0
```

```
            ei                        ;habilita as interrupções
            call    cls               ;limpa realmente a tela
            call    initxt            ;restabelece os desenhos
                                      ;originais
            ret                       ;volta para o BASIC

espera
            ld      hl,#1000          ;altere este valor para uma
                                      ;velocidade diferente
espera1     dec     hl                ;decrementa hl
            ld      a,h               ;e testa para
            or      l                 ;ver se hl=0
            jr      nz,espera1        ;Se não for, volta para
                                      ;espera1
            ret                       ;Se for, volta para a rot.
                                      ;principal

lelinha
            in      a,(#98)           ;lê o carac. da VRAM
            ld      (hl),a            ;salva-o no buffer
            inc     hl                ;incrementa o ponteiro
            djnz    lelinha           ;prepara a próxima leitura
            ret                       ;retorna

esclinha
            ld      a,(hl)            ;lê o carac. do buffer
            out     (#98),a           ;escreve-o na VRAM
            inc     hl                ;incrementa o ponteiro
            djnz    esclinha          ;prepara a próxima escrita
            ret                       ;retorna

setvdprd
            ld      a,(versao)        ;obtém a versão do MSX
            or      a                 ;é MSX1?
            jr      z,rdvram1         ;Sim, vai para rdvram1
            xor     a                 ;Não, inicializa o VDP
            out     (#99),a           ;do MSX2
            ld      a,#8e
            out     (#99),a           ;
rdvram1     ld      a,l               ;informa ao
            out     (#99),a           ;VDP o endereço na
            ld      a,h               ;VRAM onde será
            and     #3f               ;lido o dado
            out     (#99),a           ;
            ret

setvdpwt
```

```
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,wtvram1       ;Sim, vai para wtvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a
wtvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a         ;
                ret


endtab                  defw    #0000
buffer                  defs    9

fim                     equ     $
```

Listagem em linhas DATA do código-objeto do programa para limpar a tela com efeito persiana:

```
10 FOR A%=&HD000 TO &HD0A1
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG13.BIN",&HD000,&HD0A1
100 DATA F3,2A,B7,F3,3A,2D,00,B7,28,08,3A,B0,F3,FE,29,38
110 DATA 01,29,22,96,D0,06,08,C5,06,00,2A,96,D0,11,08,00
120 DATA C5,D5,E5,CD,68,D0,06,08,21,98,D0,CD,5A,D0,36,00
130 DATA E1,E5,CD,7E,D0,06,08,21,99,D0,CD,61,D0,E1,D1,C1
140 DATA 19,10,DD,CD,51,D0,C1,10,CE,FB,CD,C3,00,CD,6C,00
150 DATA C9,21,00,10,2B,7C,B5,20,FB,C9,DB,98,77,23,10,FA
160 DATA C9,7E,D3,98,23,10,FA,C9,3A,2D,00,B7,28,07,AF,D3
170 DATA 99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,C9,3A,2D
180 DATA 00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6
190 DATA 3F,F6,40,D3,99,C9,00,00,00,00,00,00,00,00,00,00
200 DATA 00,99
```

Listagem do programa de teste:

```
10 CLS:WIDTH 40
20 BLOAD"PROG9.BIN":FILES:DEFUSR=&HD000:A=USR(40)
```

```
30 BLOAD"PROG10.BIN":FILES:DEFUSR=&HD000:A=USR(40)
40 BLOAD"PROG11.BIN":FILES:DEFUSR=&HD000:A=USR(24)
50 BLOAD"PROG12.BIN":FILES:DEFUSR=&HD000:A=USR(24)
60 BLOAD"PROG13.BIN":FILES:DEFUSR=&HD000:A=USR(0)
```

Rode o programa de teste acima e "curta" o efeito dos cinco novos **CLS** que você pode implementar nos seus programas para dar aquele toque pessoal. Observe que nos quatro primeiros programas (LIMPAR rotando a tela) o argumento indica o número de linhas/colunas a LIMPAR. Alterando este argumento, você poderá LIMPAR apenas parte da tela, preservando as informações noutra parte.

A TELA GRÁFICA DE ALTA RESOLUÇÃO (SCREEN 2)

Com os programas/rotinas anteriores encerramos a nossa exploração das características do modo texto do MSX. A partir de agora, vamos nos dedicar ao entendimento do funcionamento do modo gráfico de alta resolução (**SCREEN 2** do BASIC). As rotinas apresentadas neste livro são básicas, possuindo uma finalidade puramente introdutória. Acontece que as principais características deste modo são a cor e a animação, principalmente nos jogos. Vamos então aos fundamentos deste modo de tela. Como já sabemos, todos os modos de tela apresentam pelo menos duas tabelas: a tabela de nomes, que contém um mapa do que aparece e como aparece na tela, e a tabela de padrões, que contém os desenhos de todos os caracteres que aparecem na tela. O modo gráfico de alta resolução apresenta mais três tabelas: a tabela de cores, que armazena os atributos de cor, a tabela de atributos dos sprites, que armazena os dados dos sprites como posição e cor, e a tabela de padrões dos sprites, que contém os desenhos de todos eles. Antes de passar para o estudo de cada uma destas tabelas, vamos entender o mecanismo do modo gráfico de alta resolução.

GERENCIAMENTO DA TELA GRÁFICA

A tabela de padrões deste modo apresenta uma extensão de 6Kb, ao contrário dos 2Kb da mesma tabela no modo texto. "Quer dizer que, em vez de 256 caracteres, temos 256*3=768 caracteres?" Não exatamente. Na verdade, a tela no modo gráfico de alta resolução pode ser mentalizada como um conjunto de três telas distintas, ou seja, como se a tela real (aquela que vemos) estivesse dividida em três partes horizontais iguais. Esta divisão tem uma vantagem: podemos definir três conjuntos diferentes de 256 caracteres. Vejamos o caso de um jogo da série Nemesis, da Konami. Neste tipo de jogo apresentam-se três planos de ação: um superior (geralmente a parte de cima de um túnel), um mediano (geralmente um fundo preto com estrelas e asteróides), e um inferior (geralmente a parte de baixo do túnel). Acho que agora você já está entendendo por que os jogos com maior riqueza de detalhes têm a ação se processando lateralmente. "Mas, e se a ação do jogo se processar verticalmente?" Bem, nada impede que a ação se desenrole no sentido vertical, só que, neste caso, os três conjuntos de caracteres terão que ser rigorosamente iguais, ou seja, só poderemos ter 256 caracteres. Ob-

serve que o termo caractere neste modo gráfico faz referência a um desenho criado pelo usuário (ou programador do jogo), e não a um caractere pertencente ao conjunto ASCII, como por exemplo a letra A. No modo gráfico de alta resolução (SCREEN 2), você não é capaz de imprimir um caractere na tela diretamente. Para fazer isso, você terá de abrir um canal (instrução **OPEN "GRP:"** do BASIC) para o modo gráfico e imprimir na tela usando esse canal. Tudo isso se torna necessário porque o modo gráfico não possui desenhos predefinidos para os caracteres, como acontece com o modo texto. Como na tabela de padrões do modo texto, a tabela de padrões do modo gráfico armazena os desenhos dos caracteres em matrizes 8x8 bits. Como o modo gráfico apresenta uma resolução horizontal de 256 pontos (pixéis) e vertical de 192 pontos, e já que o desenho de cada caractere gasta 8 pixéis na vertical por 8 pixéis na horizontal, podemos ter 192/8=24 linhas e 256/8=32 colunas. Desta forma, a tabela de nomes deverá ter um comprimento de 24*32=768 bytes. A vantagem do modo gráfico sobre o modo texto está além de poder receber a definição de três conjuntos distintos de 256 caracteres; está na possibilidade de atribuir até 16 cores para cada caractere. "Como assim?" A tabela de cores (inexistente no modo texto) tem exatamente o mesmo tamanho da tabela de padrões, de modo que, para cada byte na tabela de padrões, existe um byte associado na tabela de cores que define duas cores para o byte correspondente na outra tabela de padrões: uma cor para os bits iguais a 1 e outra para os bits iguais a 0. Como o desenho de cada caractere gasta 8 bytes, temos 8*2=16 cores possíveis para um caractere. Como no MSX 1 (e no MSX 2 no modo gráfico 2) só temos 16 cores, nos bastam 4 bits (2^4=16) para representar todas as cores. Tendo em vista que cada byte se constitui de 8 bits, o VDP obtém os 4 bits inferiores (nas posições de 0 a 3) para representar a cor dos bits iguais a zero do byte associado na tabela de padrões, e os 4 bits superiores (nas posições de 4 a 7) para representar a cor dos bits iguais a 1 do byte associado na tabela de padrões. Para entender melhor como funciona a tabela de padrões e a tabela de cores, digite e execute o programa em BASIC abaixo:

```
10 SCREEN 2
20 REM A LINHA 40 DEFINE O 1o. BYTE DO DESENHO DO
30 REM SEGUNDO CARACTERE
40 VPOKE &H8,&B10101010
50 REM A LINHA 70 DEFINE O 2o. BYTE DO DESENHO DO
60 REM SEGUNDO CARACTERE
70 VPOKE &H9,&B01010101
80 REM AS LINHAS 100 E 110 DEFINEM AS CORES ASSOCIADAS AOS
90 REM BYTES &H8 E &H9 DA TABELA DE PADRÕES
100 VPOKE &H2008,&HF1:REM BITS 1=BRANCO BITS 0=PRETO
110 VPOKE &H2009,&H1F:REM BITS 1=PRETO BITS 0=BRANCO
120 GOTO 120
```

Você pode estar estranhando os endereços usados. Acontece que a tabela de padrões se inicia no endereço #0000 da VRAM e termina no endereço #17FF; já a tabela de cores se inicia no endereço #2000 da VRAM e termina no endereço #37FF. Mas, após executar o programa, o que você vê na tela? Se você estiver usando um monitor de boa qualidade, verá que a tela apresenta duas linhas paralelas exatamente iguais formadas por quatro pontos intercalados. Por que isso, se os bytes colocados nos endereços #0008 e #0009 (os dois primeiros

bytes do desenho do segundo caractere) são diferentes? A resposta está na cor dada aos bits (pixéis) zero e um dos dois bytes. Observe que ao primeiro byte do desenho foi dada a cor branca (&hf) para os bits iguais a 1 e preta (&h1) para os bits iguais a 0, e ao segundo byte foi dada a cor preta (&h1) para os bits iguais a 1 e branca (&hf) para os bits iguais a 0. Devido a esta inversão nas cores é que ocorre a ilusão de que o primeiro e o segundo bytes do desenho do segundo caractere na tabela de padrões são iguais. Modifique a linha 110 para:

110 VPOKE &H2009,&HF1:REM BITS 1=BRANCO BITS 0=PRETO

Execute novamente o programa e veja que agora as duas linhas já não são mais iguais. Bem, acho que já chega de teoria, não? Vamos então à prática. Vou agora apresentar duas rotinas que fazem exatamente a mesma rotação de tela (para a esquerda e para a direita) que a apresentada para o modo texto. Você vai observar que continuaremos trabalhando com a tabela de nomes cujo endereço inicial na VRAM é tirado da variável do sistema **GRPNAM** (veja a Tabela 1.2). Vamos então às listagens dos dois programas que rotam a tela gráfica para a esquerda e para a direita respectivamente.

O PROGRAMA QUE ROTA A TELA GRÁFICA PARA A ESQUERDA

Listagem em assembly Z-80 do código-fonte do programa para rotar a tela gráfica para a esquerda:

```
;programa para rotar a tela gráfica para a esquerda
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG14.BIN=PROG14.GEN
;
;onde PROG14.GEN é o nome do arquivo-texto com esta
;listagem

versao          equ     #002d
linlen          equ     #f3b0
grpnam          equ     #f3c7
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7
scrmod          equ     #fcaf

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
```

```
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(scrmod)      ;o MSX está no modo
        cp      #02             ;gráfico?
        ret     nz              ;Não, retorna ao BASIC
        ld      a,(valtyp)      ;verifica parâmetro
        cp      #02             ;é inteiro?
        ret     nz              ;Não, volta para o BASIC
        ld      a,(argusr)      ;a=número de rotações
        push    af              ;salva número de rotações
        ld      a,32            ;a=32 colunas
        ld      (numcol),a      ;coloca o núm. de colunas
                                ;em numcol
        pop     af              ;recupera o núm. de rotações
        di                      ;desabilita as interrupções
        ld      b,a             ;b=número de rotações
loop1   push    bc              ;salva contador externo
        ld      hl,(grpnam)     ;hl=end. tabela de nomes
        ld      a,(numcol)      ;a=núm. de colunas
        ld      e,a             ;de=núm. de colunas
        ld      d,#00           ;
        ld      b,24            ;b=24 linhas na tela
loop2   push    bc              ;salva contador intermediário
        call    setvdprd        ;prepara o VDP para leitura
        ld      a,(numcol)      ;a=núm. de colunas
        push    de              ;salva de
        push    hl              ;salva hl
        ld      hl,bufferlinha  ;hl aponta para o buffer
        ld      b,a             ;b=núm. de colunas
        call    lelinha         ;lê uma linha
        ld      a,(numcol)      ;a=núm. de colunas
        ld      hl,bufferlinha  ;hl aponta para o buffer
        ld      e,a             ;de=núm. de colunas
        ld      d,#00           ;
        ld      a,(hl)          ;a=primeiro carac. da linha
        add     hl,de           ;hl=aponta para o fim da
                                ;linha+1
        ld      (hl),a          ;coloca o prim. carac. na
                                ;última posição
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        call    setvdpwt        ;prepara o VDP para escrita
```

```
                push    de                  ;salva de
                push    hl                  ;salva hl
                ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
                ld      a,(numcol)          ;a=núm. de colunas
                ld      b,a                 ;b=núm. de colunas
                call    esclinha            ;envia a linha já rotada
                pop     hl                  ;recupera hl
                pop     de                  ;recupera de
                add     hl,de               ;hl aponta para a próx. linha
                pop     bc                  ;recupera bc
                djnz    loop2               ;repete para as demais linhas
                pop     bc                  ;recupera bc
                djnz    loop1               ;repete até terminar todas
                                            ;as rotações
                ei                          ;habilita as interrupções
                ret                         ;retorna ao BASIC

lelinha
                in      a,(#98)             ;lê o carac. da VRAM
                ld      (hl),a              ;salva-o no buffer
                inc     hl                  ;incrementa o ponteiro
                djnz    lelinha             ;prepara a próxima leitura
                ret                         ;retorna

esclinha
                ld      a,(hl)              ;lê o carac. do buffer
                out     (#98),a             ;escreve-o na VRAM
                inc     hl                  ;incrementa o ponteiro
                djnz    esclinha            ;prepara a próxima escrita
                ret                         ;retorna

setvdprd
                ld      a,(versao)          ;obtém a versão do MSX
                or      a                   ;é MSX1?
                jr      z,rdvram1           ;Sim, vai para rdvram1
                xor     a                   ;Não, inicializa o VDP
                out     (#99),a             ;do MSX2
                ld      a,#8e
                out     (#99),a             ;
rdvram1         ld      a,l                 ;informa ao
                out     (#99),a             ;VDP o endereço na
                ld      a,h                 ;VRAM onde será
                and     #3f                 ;lido o dado
                out     (#99),a             ;
                ret
```

```
setvdpwt
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,wtvram1       ;Sim, vai para wtvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a
wtvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde o
            and     #3f             ;dado será
            or      #40             ;gravado
            out     (#99),a         ;
            ret


bufferlinha         defs    33
numcol              defb    #00

fim                 equ     $
```

Listagem em linhas DATA do código-objeto do programa para rotar a tela gráfica para a esquerda:

```
10 FOR A%=&HD000 TO &HD0BA
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG14.BIN",&HD000,&HD0BA
100 DATA 3A,AF,FC,FE,02,C0,3A,63,F6,FE,02,C0,3A,F8,F7,F5
110 DATA 3E,20,32,B9,D0,F1,F3,47,C5,2A,C7,F3,3A,B9,D0,5F
120 DATA 16,00,06,18,C5,CD,6A,D0,3A,B9,D0,D5,E5,21,98,D0
130 DATA 47,CD,5C,D0,3A,B9,D0,21,98,D0,5F,16,00,7E,19,77
140 DATA E1,D1,CD,80,D0,D5,E5,21,99,D0,3A,B9,D0,47,CD,63
150 DATA D0,E1,D1,19,C1,10,CD,C1,10,BE,FB,C9,DB,98,77,23
160 DATA 10,FA,C9,7E,D3,98,23,10,FA,C9,3A,2D,00,B7,28,07
170 DATA AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,C9
180 DATA 3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99
190 DATA 7C,E6,3F,F6,40,D3,99,C9,00,00,00,00,00,00,00,00
200 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
210 DATA 00,00,00,00,00,00,00,00,00,00,00
```

Listagem do programa de teste:

```
100 BLOAD"PROG14.BIN"
110 SCREEN 2
120 LINE (10,10)-(40,40),15,BF
130 LINE (215,150)-(245,180),12,BF
140 LINE (215,10)-(245,40),13,BF
150 LINE (10,150)-(40,180),14,BF
160 CIRCLE (128,95),80,3
170 PAINT (128,95),3
180 DEFUSR=&HD000
190 IF INKEY$="" THEN 190 ELSE A=USR(1):GOTO 190
```

O PROGRAMA QUE ROTA A TELA GRÁFICA PARA A DIREITA

Listagem em assembly Z-80 do código-fonte do programa para rotar a tela gráfica para a direita:

```
;programa para rotar a tela gráfica para a direita
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG15.BIN=PROG15.GEN
;
;onde PROG15.GEN é o nome do arquivo-texto com esta
;listagem

versao          equ     #002d
linlen          equ     #f3b0
grpnam          equ     #f3c7
valtyp          equ     #f663
argusr          equ     #f7f8
txtcgp          equ     #f3b7
scrmod          equ     #fcaf

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000
```

```
inicio
                ld      a,(scrmod)          ;o MSX está no modo
                cp      #02                 ;gráfico?
                ret     nz                  ;Não, retorna ao BASIC
                ld      a,(valtyp)          ;verifica parâmetro
                cp      #02                 ;é inteiro?
                ret     nz                  ;Não, volta para o BASIC
                ld      a,(argusr)          ;a=número de rotações
                push    af                  ;salva número de rotações
                ld      a,32                ;a=32 colunas
                ld      (numcol),a          ;coloca o núm. de colunas em
                                            ;numcol
                pop     af                  ;recupera o núm. de rotações
                di                          ;desabilita as interrupções
                ld      b,a                 ;b=número de rotações
loop1           push    bc                  ;salva contador externo
                ld      hl,(grpnam)         ;hl=end. tabela de nomes
                ld      a,(numcol)          ;a=núm. de colunas
                ld      e,a                 ;de=núm. de colunas
                ld      d,#00               ;
                ld      b,24                ;b=24 linhas na tela
loop2           push    bc                  ;salva contador intermediário
                call    setvdprd            ;prepara o VDP para leitura
                ld      a,(numcol)          ;a=núm. de colunas
                push    de                  ;salva de
                push    hl                  ;salva hl
                ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
                ld      b,a                 ;b=núm. de colunas
                call    lelinha             ;lê uma linha
                ld      a,(numcol)          ;a=núm. de colunas
                ld      hl,bufferlinha      ;hl aponta para o buffer
                ld      e,a                 ;de=núm. de colunas
                ld      d,#00               ;
                add     hl,de               ;hl aponta para o últ. carac.
                ld      a,(hl)              ;a=último carac. da linha
                ld      hl,bufferlinha      ;hl aponta para o buffer
                ld      (hl),a              ;coloca o últ. carac. na
                                            ;primeira posição
                pop     hl                  ;recupera hl
                pop     de                  ;recupera de
                call    setvdpwt            ;prepara o VDP para escrita
                push    de                  ;salva de
                push    hl                  ;salva hl
                ld      hl,bufferlinha      ;hl aponta para o buffer
                ld      a,(numcol)          ;a=núm. de colunas
                ld      b,a                 ;b=núm. de colunas
                call    esclinha            ;envia a linha já rotada
```

```
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        add     hl,de           ;hl aponta para a próx. linha
        pop     bc              ;recupera bc
        djnz    loop2           ;repete para as demais linhas
        pop     bc              ;recupera bc
        djnz    loop1           ;repete até terminar todas
                                ;as rotações
        ei                      ;habilita as interrupções
        ret                     ;retorna ao BASIC

lelinha
        in      a,(#98)         ;lê o carac. da VRAM
        ld      (hl),a          ;salva-o no buffer
        inc     hl              ;incrementa o ponteiro
        djnz    lelinha         ;prepara a próxima leitura
        ret                     ;retorna

esclinha
        ld      a,(hl)          ;lê o carac. do buffer
        out     (#98),a         ;escreve-o na VRAM
        inc     hl              ;incrementa o ponteiro
        djnz    esclinha        ;prepara a próxima escrita
        ret                     ;retorna

setvdprd
        ld      a,(versao)      ;obtém a versão do MSX
        or      a               ;é MSX1?
        jr      z,rdvram1       ;Sim, vai para rdvram1
        xor     a               ;Não, inicializa o VDP
        out     (#99),a         ;do MSX2
        ld      a,#8e
        out     (#99),a         ;
rdvram1 ld      a,l             ;informa ao
        out     (#99),a         ;VDP o endereço na
        ld      a,h             ;VRAM onde será
        and     #3f             ;lido o dado
        out     (#99),a         ;
        ret

setvdpwt
        ld      a,(versao)      ;obtém a versão do MSX
        or      a               ;é MSX1?
        jr      z,wtvram1       ;Sim, vai para wtvram1
        xor     a               ;Não, inicializa o VDP
        out     (#99),a         ;do MSX2
```

```
                ld      a,#8e
                out     (#99),a
wtvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a         ;
                ret


bufferlinha             defs    33
numcol                  defb    #00

fim                     equ     $
```

Listagem em linhas DATA do código-objeto do programa para rotar a tela gráfica para a direita:

```
10 FOR A%=&HD000 TO &HD0BD
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG15.BIN",&HD000,&HD0BD
100 DATA 3A,AF,FC,FE,02,C0,3A,63,F6,FE,02,C0,3A,F8,F7,F5
110 DATA 3E,20,32,BC,D0,F1,F3,47,C5,2A,C7,F3,3A,BC,D0,5F
120 DATA 16,00,06,18,C5,CD,6D,D0,3A,BC,D0,D5,E5,21,9C,D0
130 DATA 47,CD,5F,D0,3A,BC,D0,21,9B,D0,5F,16,00,19,7E,21
140 DATA 9B,D0,77,E1,D1,CD,83,D0,D5,E5,21,9B,D0,3A,BC,D0
150 DATA 47,CD,66,D0,E1,D1,19,C1,10,CA,C1,10,BB,FB,C9,DB
160 DATA 98,77,23,10,FA,C9,7E,D3,98,23,10,FA,C9,3A,2D,00
170 DATA B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F
180 DATA D3,99,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99
190 DATA 7D,D3,99,7C,E6,3F,F6,40,D3,99,C9,00,00,00,00,00
200 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
210 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00
```

Listagem do programa de teste:

```
100 BLOAD"PROG15.BIN"
110 SCREEN 2
120 LINE (10,10)-(40,40),15,BF
130 LINE (215,150)-(245,180),12,BF
140 LINE (215,10)-(245,40),13,BF
150 LINE (10,150)-(40,180),14,BF
```

```
160 CIRCLE (128,95),80,3
170 PAINT (128,95),3
180 DEFUSR=&HD000
190 IF INKEY$="" THEN 190 ELSE A=USR(1):GOTO 190
```

COMENTÁRIOS SOBRE OS PROGRAMAS DE ROTAÇÃO DA TELA GRÁFICA

Antes de executar os programas de teste, repare que existem alguns comandos em BASIC para a construção de figuras gráficas em diversas cores. Esta etapa se torna necessária para podermos preencher as tabelas de padrões e de cores, e assim usarmos a tabela de nomes. O preenchimento das tabelas de padrões e de cores se torna necessário devido ao fato de que, quando se submete o comando **SCREEN 2** ao interpretador BASIC do MSX, ocorre uma inicialização da VRAM, o que implica afirmar que a tabela de padrões é preenchida com zeros (sem pixéis) e a tabela de cores com uns (#01=cor transparente (#0) para os bits iguais a 1 e cor preta (#1) para os bits iguais a 0). Geralmente se diz que a cor dada aos bits iguais a 1 na tabela de padrões é a cor de frente, e a cor dada aos bits iguais a 0 na tabela de padrões é a cor de fundo. Seguindo esta nomenclatura, após a inicialização feita pelo comando SCREEN 2 podemos dizer que a cor de frente é transparente e a cor de fundo é preta. Vale também observar que a cor de fundo pode variar entre os modelos de MSX, ou seja, a tabela de cores pode ser inicializada com uma cor de fundo igual à azul, ao invés de preta. Uma vez preenchidas as tabelas de padrões e de cores, podemos aplicar o nosso algoritmo para a rotação da tela. O algoritmo usado para a rotação de uma tela gráfica é exatamente o mesmo que o usado para a rotação de uma tela no modo texto. As únicas exceções estão no tamanho da tabela de nomes e no respectivo endereço inicial na VRAM. Basta acompanhar os comentários feitos nas listagens do código-fonte para tirar qualquer dúvida.

Ao executar os programas de teste, observe que o argumento da função **USR** é 1, ou seja, só é feita uma rotação (de apenas uma coluna) por vez. A linha 190 garante a realização de cada rotação enquanto você estiver pressionando uma tecla qualquer. Se você desejar fazer uma rotação completa da tela, deverá entrar com 32 (o número de colunas numa tela gráfica) como argumento da função USR.

OS SPRITES NA TELA DE ALTA RESOLUÇÃO

Entre as diversas cartas que recebi, muitos leitores me perguntaram como movimentar um sprite pela tela usando somente as teclas do cursor. Antes de passar à rotina que realiza tal tarefa, vamos ver um pouco de teoria. No modo gráfico 2 existem duas tabelas dedicadas aos sprites: a tabela de padrões dos sprites, que contém os desenhos dos sprites, e a tabela de atributos dos sprites, que contém as coordenadas (x,y), a cor dos bits iguais a 1 no desenho e o plano (0 a 31) do sprite. Como você pode perceber, a tabela de padrões dos sprites é uma espécie de fusão das tabelas de nomes e de cores normais. Os endereços das duas tabelas dos sprites na VRAM são fornecidos pelas variáveis do sistema **GRPATR** e **GRPPAT** (consulte a Tabela 1.2).

A rotina em si é bastante simples, apesar de empregar uma rotina para leitura direta do teclado. Não se preocupe com esta última rotina, pois vamos estudá-la em detalhes numa próxima oportunidade. Vamos então à listagem do código-fonte do programa para movimentar sprites pela tela.

O PROGRAMA QUE IMPLEMENTA O CONTROLE SOBRE OS SPRITES

Listagem em assembly Z-80 do código-fonte do programa para movimentar sprites pela tela:

```
;programa para movimentar sprites pela tela
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG16.BIN=PROG16.GEN
;
;onde PROG16.GEN é o nome do arquivo-texto com esta
;listagem

versao          equu    #002d
grppat          equu    #f3cf
grpatr          equu    #f3cd
scrmod          equu    #fcaf

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(scrmod)      ;o MSX está no modo
        cp      #02             ;gráfico?
        ret     nz              ;Não, retorna ao BASIC
        di                      ;desabilita as interrupções
        ld      hl,(grppat)     ;hl aponta para a tab. de
                                ;padrões
        ld      de,desenho      ;de aponta para o desenho
        ld      b,#08           ;b=8 bytes que formam o
                                ;desenho
loop1   ld      a,(de)          ;este loop envia o desenho
        call    wtvram          ;para a tabela de padrões
```

```
         inc     de              ;dos sprites na RAM de vídeo
         inc     hl              ;(VRAM)
         djnz    loop1           ;
         ld      hl,(grpatr)     ;hl aponta para a tab. de
                                 ;atributos
         ld      de,atributos    ;de aponta para os atributos
         ld      b,#04           ;b=4=núm. de bytes dos
                                 ;atributos
loop2    ld      a,(de)          ;este loop envia os atributos
         call    wtvram          ;para a tabela de atributos
         inc     de              ;dos sprites na RAM de vídeo
         inc     hl              ;(VRAM)
         djnz    loop2           ;
loopprin
         call    letecla         ;lê as teclas do cursor e
                                 ;barra de espaço
         push    af              ;salva o valor lido
         bit     7,a             ;a seta para direita foi
                                 ;pressionada?
         call    z,direita       ;Sim, chama a rotina direita
         pop     af              ;recupera o valor lido
         push    af              ;torna a salvar
         bit     4,a             ;a seta para esquerda foi
                                 ;pressionada?
         call    z,esquerda      ;Sim, chama a rotina
                                 ;esquerda
         pop     af              ;recupera o valor lido
         push    af              ;torna a salvar
         bit     5,a             ;a seta para cima foi
                                 ;pressionada?
         call    z,cima          ;Sim, chama a rotina cima
         pop     af              ;recupera o valor lido
         push    af              ;torna a salvar
         bit     6,a             ;a seta para baixo foi
                                 ;pressionada?
         call    z,baixo         ;Sim, chama a rotina baixo
         pop     af              ;recupera o valor lido
         bit     0,a             ;a barra de espaço foi
                                 ;pressionada?
         jr      z,basic         ;Sim, prepara a volta ao
                                 ;BASIC
         call    espera          ;Não, espera para diminuir a
                                 ;velocidade do movimento
         jr      loopprin        ;fecha o loop
basic
```

```
          ei                      ;habilita as interrupções
          ret                     ;retorna ao BASIC

letecla
          in       a,(#aa)        ;prepara o ppi para ler
          and      #f0            ;a linha (#08) com as
          or       #08            ;informações sobre as
          out      (#aa),a        ;teclas do cursor
          in       a,(#a9)        ;lê a coluna selecionada
          ret                     ;volta

espera
          ld       hl,#1000       ;provoca um retardo
loopesp   dec      hl             ;baseado em sucessivos
          ld       a,h            ;decrementos no valor de
          or       l              ;hl
          jr       nz,loopesp     ;
          ret                     ;

direita
          ld       hl,(grpatr)    ;hl aponta para o atributo X
                                  ;na VRAM
          inc      hl
          call     rdvram         ;lê a posição x do sprite
          inc      a              ;incrementa-a
          call     wtvram         ;informa ao VDP a nova
                                  ;posição
          ret                     ;retorna

esquerda
          ld       hl,(grpatr)    ;hl aponta para o atributo X
                                  ;na VRAM
          inc      hl
          call     rdvram         ;lê a posição x do sprite
          dec      a              ;decrementa-a
          call     wtvram         ;informa ao VDP a nova
                                  ;posição
          ret                     ;retorna

cima
          ld       hl,(grpatr)    ;hl aponta para o atributo Y
                                  ;na VRAM
          call     rdvram         ;lê a posição y do sprite
          or       a              ;é zero?
          jr       z,cima1        ;Sim, vai para cima1
          dec      a              ;Não, decrementa y (sobe o
                                  ;sprite)
```

```
            jr      cima2           ;vai para cima2
cima1       ld      a,191           ;coloca o sprite na ultima
                                    ;linha
cima2       call    wtvram          ;informa ao VDP a nova
                                    ;posição
            ret                     ;retorna

baixo
            ld      hl,(grpatr)     ;hl aponta para o atributo Y
                                    ;na VRAM
            call    rdvram          ;lê a posição y do sprite
            cp      191             ;já está na última linha?
            jr      z,baixo1        ;Sim, vai para baixo1
            inc     a               ;Não, incrementa y (desce o
                                    ;sprite)
            jr      baixo2          ;vai para baixo2
baixo1      xor     a               ;a=0=primeira linha
baixo2      call    wtvram          ;informa ao VDP a nova
                                    ;posição
            ret                     ;retorna


rdvram
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,rdvram1       ;Sim, vai para rdvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a         ;
rdvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde será
            and     #3f             ;lido o dado
            out     (#99),a         ;
            ex      (sp),hl         ;demora para
            ex      (sp),hl         ;sincronização
            in      a,(#98)         ;lê o dado na VRAM
            ret

wtvram
            push    af              ;salva dado a ser gravado
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,wtvram1       ;Sim, vai para wtvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
```

```
                ld      a,#8e
                out     (#99),a
wtvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a         ;
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                pop     af              ;recupera o dado
                out     (#98),a         ;grava o dado
                ret

;Tabela de 8 bytes com o desenho do boneco

desenho
                        defb    %00111000
                        defb    %00101000
                        defb    %00111000
                        defb    %00010000
                        defb    %00111000
                        defb    %01010100
                        defb    %00101000
                        defb    %01000100

;Tabela dos atributos do sprite

atributos
y                       defb    86              ;coordenadas do ponto
x                       defb    128             ;central da tela
modelo                  defb    0               ;primeiro plano
cor                     defb    15              ;cor branca para o sprite

fim                     equ     $
```

Listagem em linhas DATA do código-objeto do programa para movimentar sprites p tela:

```
10 FOR A%=&HD000 TO &HD0E5
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG16.BIN",&HD000,&HD0E5
```

```
100 DATA 3A,AF,FC,FE,02,C0,F3,2A,CF,F3,11,D9,D0,06,08,1A
110 DATA CD,BB,D0,13,23,10,F8,2A,CD,F3,11,E1,D0,06,04,1A
120 DATA CD,BB,D0,13,23,10,F8,CD,51,D0,F5,CB,7F,CC,65,D0
130 DATA F1,F5,CB,67,CC,71,D0,F1,F5,CB,6F,CC,7D,D0,F1,F5
140 DATA CB,77,CC,8F,D0,F1,CB,47,28,05,CD,5C,D0,18,D8,FB
150 DATA C9,DB,AA,E6,F0,F6,08,D3,AA,DB,A9,C9,21,00,10,2B
160 DATA 7C,B5,20,FB,C9,2A,CD,F3,23,CD,A1,D0,3C,CD,BB,D0
170 DATA C9,2A,CD,F3,23,CD,A1,D0,3D,CD,BB,D0,C9,2A,CD,F3
180 DATA CD,A1,D0,B7,28,03,3D,18,02,3E,BF,CD,BB,D0,C9,2A
190 DATA CD,F3,CD,A1,D0,FE,BF,28,03,3C,18,01,AF,CD,BB,D0
200 DATA C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3
210 DATA 99,7C,E6,3F,D3,99,E3,E3,DB,98,C9,F5,3A,2D,00,B7
220 DATA 28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6
230 DATA 40,D3,99,E3,E3,F1,D3,98,C9,38,28,38,10,38,54,28
240 DATA 44,56,80,00,0F,00
```

Listagem do programa de teste:

```
100 BLOAD"PROG16.BIN"
110 SCREEN 2
120 LINE (10,10)-(40,40),15,BF
130 LINE (215,150)-(245,180),12,BF
140 LINE (215,10)-(245,40),13,BF
150 LINE (10,150)-(40,180),14,BF
160 CIRCLE (128,95),80,3
170 PAINT (128,95),3
180 DEFUSR=&HD000
190 A=USR(0)
```

COMENTÁRIOS SOBRE O PROGRAMA QUE IMPLEMENTA O CONTROLE SOBRE OS SPRITES

Observe que o programa de teste desenha e pinta algumas figuras gráficas antes de chamar a rotina em assembly para a criação e movimentação do sprite. O programa, tal como os anteriores já apresentados, é bem simples, bastando apenas acompanhar os comentários na listagem do código-fonte para um perfeito entendimento. Se você examinar a listagem do código-fonte, verá que a saída da rotina de movimentação do sprite na tela é determinada pelo pressionamento da barra de espaço. Desta forma, as teclas do cursor movimentam o sprite nas diversas direções (tente, por exemplo, pressionar as teclas da seta para cima e para a direita ao mesmo tempo), e a barra de espaço encerra a movimentação.

USANDO CARACTERES EM VEZ DE SPRITES NA TELA GRÁFICA

Você já deve ter percebido que nos jogos animados nem todos os objetos são sprites. Vejamos, por exemplo, o caso do jogo ARMY MOVIES. Neste jogo, temos um jipe que se movimenta na horizontal da esquerda para a direita. Ao que tudo indica, o jipe é formado por um conjunto de seis sprites (três em cima e três em baixo). Entretanto, se repararmos bem, veremos que, à medida que o jipe se movimenta, a cor do fundo (a cor das montanhas, no caso) adquire a mesma cor do jipe. Se o jipe fosse realmente constituído por sprites, esse efeito de "borramento" não ocorreria. O que forma, então, o jipe? A resposta é simples: o jipe é formado por um conjunto de seis caracteres (lembre-se que caractere, em tela gráfica, não quer dizer caractere em ASCII). Pode parecer um retrocesso usar conjuntos de caracteres ao invés de sprites, mas acontece que a maioria dos jogos europeus para o MSX é uma adaptação dos mesmos jogos feitos para o ZX Spectrum. Como o Spectrum (no BRASIL, recebeu o nome de TK-90X) não possui o recurso de sprites, o jeito foi simulá-los usando a redefinição dos caracteres. Para se ter uma idéia de como esta técnica evoluiu, basta dizer que a famosa Filmation II é uma descendente da redefinição dos caracteres. A técnica da sobreposição dos caracteres redefinidos é bem simples. Vamos acompanhar um exemplo: suponha que desejemos desenhar uma mira que se movimente sobre um fundo em xadrez. Para tornar a rotina mais simples, vamos limitar o desenho e o movimento da mira ao primeiro terço (lembre-se que a tela gráfica está dividida em três) da tela. Antes de começar, vamos a um pouco de teoria. Quando você pega uma folha em branco e começa a desenhar uma casa, por exemplo, a primeira coisa que você faz é desenhar o contorno da casa com um lápis preto. Depois de desenhar o contorno é que você começa a pintar. Este é exatamente o processo que temos de empregar para desenhar na tela gráfica. Primeiro, desenha-se o fundo da tela (um xadrez, no nosso exemplo), depois o contorno do caractere (no caso, a mira); logo em seguida, desenha-se a própria mira e, por último, o caractere que corresponde ao fundo com a mira sobreposta. Como já sabemos, os desenhos dos caracteres nada mais são do que mapeamentos por bits. Vamos então aos desenhos (em bits) de cada um dos caracteres mencionados.

Caractere do fundo (xadrez):

```
10101010
01010101
10101010
01010101
10101010
01010101
10101010
01010101
```

Caractere do contorno da mira:

```
11111111
```

```
10000001
10000001
10000001
10000001
10000001
10000000
11111111
```

Caractere da mira:

```
00000000
00000000
00111100
00100100
00100100
00111100
00000000
00000000
```

Combinação dos três caracteres acima:

```
10101010
00000001
10111100
00100101
10100100
00111101
10000000
01010101
```

Observe que o desenho acima nada mais é do que uma constituição da seguinte fórmula:

caractere final = (fundo AND contorno) OR mira

Esta fórmula é o segredo de tudo. "Mas, Eduardo, qual o motivo de toda esta explicação se o MSX possui sprites?" Ocorre que você não precisa usar sprites para tudo. É muito mais simples trabalhar com a tabela de nomes do que com a tabela de atributos dos sprites. No MSX, temos dois tipos de jogos: os japoneses e os europeus. Acho que você concorda comigo quando afirmo que os jogos japoneses são de uma qualidade assustadoramente maior. Essa grande diferença se deve ao fato de os jogos japoneses serem desenvolvidos especificamente para o MSX, e não adaptados de outras máquinas, como acontece com os jogos europeus. Não obstante, os jogos japoneses também utilizam o recurso de sobreposição de caracteres, só que em escala muito reduzida. Na série Nemesis, da Konami, os canhões nas paredes dos túneis são caracteres, bem como a maioria das naves de ataque inimigas. Terminada a teoria, vamos para o exemplo prático.

O PROGRAMA QUE MOVIMENTA CARACTERES EM VEZ DE SPRITES

Listagem em assembly Z-80 do código-fonte do programa para sobreposição de caracteres gráficos:

```
;programa para sobreposição de caracteres gráficos
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG17.BIN=PROG17.GEN
;
;onde PROG17.GEN é o nome do arquivo-texto com esta
;listagem

versao          equu    #002d
grpnam          equ     #f3c7
grpcol          equ     #f3c9
grpcgp          equ     #f3cb
scrmod          equ     #fcaf

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(scrmod)      ;o MSX está no modo
        cp      #02             ;gráfico?
        ret     nz              ;Não, retorna ao BASIC
        di                      ;desabilita as interrupções
        call    transdes        ;chama a rotina transdes
        call    transcor        ;chama a rotina transcor
        call    tabnomes        ;chama a rotina tabnomes

        ld      hl,(grpnam)     ;hl aponta para a tab. de
                                ;nomes
        ld      de,32*3+16      ;de=Y=3 X=16
        add     hl,de           ;hl aponta para a pos. (X,Y)
        ld      a,#01           ;a=núm. do segundo desenho
        call    wtvram          ;coloca a mira na tela
        ld      a,16            ;a=meio da linha (coluna 16)
        ld      (coluna),a      ;salva o valor da coluna
looprin
```

```
        call    letecla         ;lê as teclas do cursor e a
                                ;barra de espaço
        push    af              ;salva o valor lido
        bit     7,a             ;a seta para direita foi
                                ;pressionada?
        call    z,direita       ;Sim, chama a rotina direita
        pop     af              ;recupera o valor lido
        push    af              ;torna a salvar
        bit     4,a             ;a seta para esquerda foi
                                ;pressionada?
        call    z,esquerda      ;Sim, chama a rotina esquerda
        pop     af              ;recupera o valor lido
        bit     0,a             ;a barra de espaço foi
                                ;pressionada?
        jr      z,basic         ;Sim, prepara a volta ao
                                ;BASIC
        call    espera          ;Não, espera para diminuir a
                                ;velocidade de processamento
        jr      loopprin        ;fecha o loop

basic
        ei                      ;habilita as interrupções
        ret                     ;retorna ao BASIC


letecla
        in      a,(#aa)         ;prepara o ppi para ler
        and     #f0             ;a linha (#08) com as
        or      #08             ;informações sobre as
        out     (#aa),a         ;teclas do cursor
        in      a,(#a9)         ;lê a coluna selecionada
        ret                     ;volta

espera
        ld      hl,#1000        ;provoca um retardo
loopesp dec     hl              ;baseado em sucessivos
        ld      a,h             ;decrementos no valor de
        or      l               ;hl
        jr      nz,loopesp      ;
        ret                     ;


direita
        ld      hl,(grpnam)     ;hl aponta para a tab. de
                                ;nomes
        ld      de,3*32         ;de=Y=3
        add     hl,de           ;hl aponta para o início da
```

```
                                             ;linha 3
              push     hl
              ld       a,(coluna)            ;obtém a coluna atual
              ld       d,#00
              ld       e,a                   ;de=coluna atual
              add      hl,de                 ;hl aponta para a pos. atual
              xor      a                     ;a=núm. do primeiro desenho
                                             ;(xadrez)
              call     wtvram                ;escreve o fundo
              pop      hl                    ;recupera o apontador para a
                                             ;linha
              ld       a,(coluna)            ;a=núm. de colunas
              cp       31                    ;chegou à última coluna?
              jr       z,direita1            ;Sim, vai para direita1
              ld       d,#00
              ld       e,a                   ;de=coluna atual
              inc      de
              add      hl,de                 ;hl aponta para a nova
                                             ;posição
              inc      a                     ;incrementa a coluna
              jr       direita2              ;vai para direita2
direita1      xor      a                     ;zera a coluna
direita2      ld       (coluna),a            ;guarda a nova coluna
              ld       a,#01                 ;a=núm. do desenho da mira
              call     wtvram                ;escreve a mira na nova
                                             ;posição
              ret                            ;retorna

esquerda
              ld       hl,(grpnam)           ;hl aponta para a tab. de
                                             ;nomes
              ld       de,3*32               ;de=Y=3
              add      hl,de                 ;hl aponta para o início da
                                             ;linha 3
              push     hl
              ld       a,(coluna)            ;obtém a coluna atual
              ld       d,#00
              ld       e,a                   ;de=coluna atual
              add      hl,de                 ;hl aponta para a posição
                                             ;atual
              xor      a                     ;a=núm. do primeiro desenho
                                             ;(xadrez)
              call     wtvram                ;escreve o fundo
              pop      hl                    ;recupera o apontador para a
                                             ;linha
              ld       a,(coluna)            ;a=núm. de colunas
```

```
                or      a               ;chegou à primeira coluna?
                jr      z,esquerda1     ;Sim, vai para esquerda1
                ld      d,#00
                ld      e,a             ;de=coluna atual
                dec     de
                add     hl,de           ;hl aponta para a nova
                                        ;posição
                dec     a               ;decrementa a coluna
                jr      esquerda2       ;vai para esquerda2
esquerda1       ld      a,31            ;a=última coluna
                ld      d,#00           ;
                ld      e,a             ;de=núm. da nova coluna
                add     hl,de           ;hl aponta para o fim da
                                        ;linha
esquerda2       ld      (coluna),a      ;guarda a nova coluna
                ld      a,#01           ;a=núm. do desenho da mira
                call    wtvram          ;escreve a mira na nova
                                        ;posição
                ret                     ;retorna


;a rotina transdes transfere os desenhos para a tabela de
;padrões

transdes
                ld      hl,(grpcgp)     ;hl aponta para a tab. de
                                        ;padrões
                ld      de,desenho1     ;de aponta para o desenho1
                                        ;(xadrez)
                ld      ix,desenho2     ;ix aponta para o desenho2
                ld      iy,desenho3     ;iy aponta para o desenho3
                push    de
                ld      b,#08           ;b=8 bytes por desenho
transdes1       ld      a,(de)          ;este loop transfere
                call    wtvram          ;o primeiro
                inc     de              ;desenho para a RAM
                inc     hl              ;de vídeo
                djnz    transdes1       ;
                pop     de
                ld      b,#08           ;b=8 bytes
transdes2       ld      a,(de)          ;a=byte do desenho de fundo
                and     (ix+#00)        ;faz um AND com a máscara da
                                        ;mira
                or      (iy+#00)        ;faz um OR com o des. da mira
                call    wtvram          ;escreve o byte
                inc     hl              ;
                inc     de              ;
```

```
                inc     ix              ;
                inc     iy              ;
                djnz    transdes2       ;repete para os 8 bytes do
                                        ;desenho
                ret                     ;retorna


;a rotina transcor fornece a cor dos desenhos

transcor
                ld      hl,(grpcol)     ;hl aponta para a tabela das
                                        ;cores
                ld      a,#f1           ;branco (f=15) para cor de
                                        ;frente e
                                        ;preto (1) para a cor de
                                        ;fundo
                ld      b,#10           ;b=16 bytes a colorir
transcor1       call    wtvram          ;loop para enviar
                inc     hl              ;a cor
                djnz    transcor1       ;
                ret                     ;retorna


;a rotina tabnomes preenche o primeiro terço da tela com
;o caractere do fundo (xadrez)

tabnomes
                ld      hl,(grpnam)     ;hl aponta para a tab. de
                                        ;nomes
                ld      b,#00           ;prepara um loop com 256
                                        ;repetições (768/3=256)
                xor     a               ;a=núm. do primeiro desenho
tabnomes1       call    wtvram          ;preenche um terço da tela
                inc     hl              ;com o primeiro desenho
                                        ;(xadrez)
                djnz    tabnomes1       ;
                ret                     ;retorna


rdvram
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,rdvram1       ;Sim, vai para rdvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
```

```
                out     (#99),a         ;
rdvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde será
                and     #3f             ;lido o dado
                out     (#99),a         ;
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                in      a,(#98)         ;lê o dado na VRAM
                ret

wtvram
                push    af              ;salva dado a ser gravado
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,wtvram1       ;Sim, vai para wtvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a
wtvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a         ;
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                pop     af              ;recupera o dado
                out     (#98),a         ;grava o dado
                ret


;Tabela de 24 bytes com os desenhos do fundo, contorno e mira

desenho1
                defb    %10101010
                defb    %01010101
                defb    %10101010
                defb    %01010101
                defb    %10101010
                defb    %01010101
                defb    %10101010
                defb    %01010101

desenho2
                defb    %11111111
```

```
                defb    %10000001
                defb    %10000001
                defb    %10000001
                defb    %10000001
                defb    %10000001
                defb    %10000001
                defb    %11111111

desenho3
                defb    %00000000
                defb    %00000000
                defb    %00111100
                defb    %00100100
                defb    %00100100
                defb    %00111100
                defb    %00000000
                defb    %00000000

coluna          defb    #00

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa para sobreposição de caracteres gráficos:

```
10 FOR A%=&HD000 TO &HD146
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG17.BIN",&HD000,&HD146
100 DATA 3A,AF,FC,FE,02,C0,F3,CD,AF,D0,CD,DE,D0,CD,EC,D0
110 DATA 2A,C7,F3,11,70,00,19,3E,01,CD,13,D1,3E,10,32,49
120 DATA D1,CD,3D,D0,F5,CB,7F,CC,51,D0,F1,F5,CB,67,CC,7E
130 DATA D0,F1,CB,47,28,05,CD,48,D0,18,E6,FB,C9,DB,AA,E6
140 DATA F0,F6,08,D3,AA,DB,A9,C9,21,00,10,2B,7C,B5,20,FB
150 DATA C9,2A,C7,F3,11,60,00,19,E5,3A,49,D1,16,00,5F,19
160 DATA AF,CD,13,D1,E1,3A,49,D1,FE,1F,28,08,16,00,5F,13
170 DATA 19,3C,18,01,AF,32,49,D1,3E,01,CD,13,D1,C9,2A,C7
180 DATA F3,11,60,00,19,E5,3A,49,D1,16,00,5F,19,AF,CD,13
190 DATA D1,E1,3A,49,D1,B7,28,08,16,00,5F,1B,19,3D,18,06
200 DATA 3E,1F,16,00,5F,19,32,49,D1,3E,01,CD,13,D1,C9,2A
210 DATA CB,F3,11,31,D1,DD,21,39,D1,FD,21,41,D1,D5,06,08
220 DATA 1A,CD,13,D1,13,23,10,F8,D1,06,08,1A,DD,A6,00,FD
230 DATA B6,00,CD,13,D1,23,13,DD,23,FD,23,10,EE,C9,2A,C9
240 DATA F3,3E,F1,06,10,CD,13,D1,23,10,FA,C9,2A,C7,F3,06
250 DATA 00,AF,CD,13,D1,23,10,FA,C9,3A,2D,00,B7,28,07,AF
```

```
260 DATA D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,E3,E3
270 DATA DB,98,C9,F5,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3
280 DATA 99,7D,D3,99,7C,E6,3F,F6,40,D3,99,E3,E3,F1,D3,98
290 DATA C9,AA,55,AA,55,AA,55,AA,55,FF,81,81,81,81,81,81
300 DATA FF,00,00,3C,24,24,3C
```

Listagem do programa de teste:

```
10 SCREEN2
20 BLOAD"PROG17.BIN"
30 DEFUSR=&HD000
40 A=USR(0)
```

COMENTÁRIOS SOBRE O PROGRAMA QUE MOVIMENTA CARACTERES

Ao executar o programa de teste, você poderá observar que a mira se desloca para a direita ou para a esquerda sem modificar o fundo da tela. É claro que este exemplo é bem simples, mas já serve para dar uma idéia de como se processa a substituição de sprites por caracteres redefinidos.

ANIMAÇÃO DE CARACTERES NA TELA DE ALTA RESOLUÇÃO

Um outro efeito que sempre me chamou a atenção é a animação de objetos. Suponha que você esteja fazendo um jogo que deve apresentar um radar. Neste caso, você teria duas opções: deixar o radar parado mostrando sempre o mesmo lado, ou fazê-lo movimentar-se sobre si mesmo. É inegável que a segunda opção exerce um apelo visual muito mais forte. Mas, como fazer tal animação? A animação, tanto faz se nos desenhos animados como no computador, consiste na troca rápida de imagens. No caso do radar, teríamos que trocar pelo menos quatro imagens: o radar de frente, de lado com o centro apontando para a direita, de trás e, por último, de lado com o centro apontando para a esquerda. Feitos os quatro desenhos, restaria transferi-los para a tabela de padrões, colocar as respectivas cores na tabela de cores e, por fim, usar a tabela de nomes para realizar a troca rápida das quatro imagens (caracteres). Parece simples, não? Na verdade, é bem simples. Neste capítulo, vou apresentar somente a rotina que faz tal animação, mas, no próximo capítulo, onde veremos as interrupções, você vai rever esta mesma rotina, só que com um efeito muito mais interessante. Bom, vamos então à listagem do programa para animação de um radar.

O PROGRAMA QUE IMPLEMENTA A ANIMAÇÃO DE CARACTERES

Listagem em assembly Z-80 do código-fonte do programa para animação em tela gráfica:

```
;programa para animação em tela gráfica
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG18.BIN=PROG18.GEN
;
;onde PROG18.GEN é o nome do arquivo-texto com esta
;listagem

versao          equ     #002d
grpnam          equ     #f3c7
grpcol          equ     #f3c9
grpcgp          equ     #f3cb
scrmod          equ     #fcaf

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

        org     #d000

inicio
        ld      a,(scrmod)      ;o MSX está no modo
        cp      #02             ;gráfico?
        ret     nz              ;Não, retorna ao BASIC
        di                      ;desabilita as interrupções
        call    transdes        ;chama a rotina transdes
        call    transcor        ;chama a rotina transcor
        call    tabnomes        ;chama a rotina tabnomes

        ld      hl,(grpnam)     ;hl aponta para a tab. de
                                ;nomes
        ld      de,32*3+16      ;de=Y=3 X=16
        add     hl,de           ;hl aponta para a posição
                                ;(X,Y)
        xor     a               ;a=núm. do primeiro desenho
        call    wtvram          ;coloca o radar na tela
        ld      e,a             ;salva o registro a no e
loopprin
```

```
                ld      a,e             ;recupera o registro a em e
                inc     a               ;incrementa o apontador do
                                        ;desenho
                and     %00000011       ;and 3 para ficar na faixa
                                        ;0..3
                ld      e,a             ;salva o registro a no e
                call    wtvram          ;escreve o carac. no vídeo
                call    letecla         ;lê as teclas do cursor e
                                        ;a barra de espaço
                bit     0,a             ;a barra de espaço foi
                                        ;pressionada?
                jr      z,basic         ;Sim, prepara a volta ao
                                        ;BASIC
                call    espera          ;Não, espera para diminuir a
                                        ;velocidade de processamento
                jr      loopprin        ;fecha o loop

basic
                ei                      ;habilita as interrupções
                ret                     ;retorna ao BASIC


letecla
                in      a,(#aa)         ;prepara o ppi para ler
                and     #f0             ;a linha (#08) com as
                or      #08             ;informações sobre as
                out     (#aa),a         ;teclas do cursor
                in      a,(#a9)         ;lê a coluna selecionada
                ret                     ;volta

espera
                push    hl              ;salva hl=apontador para a
                                        ;VRAM
                ld      hl,#8000        ;provoca um retardo
loopesp         dec     hl              ;baseado em sucessivos
                ld      a,h             ;decrementos no valor de
                or      l               ;hl
                jr      nz,loopesp      ;
                pop     hl              ;recupera hl
                ret                     ;retorna


;a rotina transdes transfere os desenhos do radar para a
;tabela de padrões

transdes
                ld      hl,(grpcgp)     ;hl aponta para a tab. de
```

```
                                                        ;padrões
                ld      de,desenho1                     ;de aponta para o desenho1
                ld      b,#08*4                         ;b=8 bytes por desenho * 4
                                                        ;desenhos
transdes1       ld      a,(de)                          ;este loop transfere
                call    wtvram                          ;todos os
                inc     de                              ;desenhos para a RAM
                inc     hl                              ;de vídeo
                djnz    transdes1                       ;
                ret                                     ;retorna

;a rotina transcor fornece a cor dos 4 desenhos
;(caracteres) transferidos para a tabela de padrões

transcor
                ld      hl,(grpcol)                     ;hl aponta para a tabela das
                                                        ;cores
                ld      a,#f1                           ;branco (f=15) para cor de
                                                        ;frente e
                                                        ;preto (1) para a cor de
                                                        ;fundo
                ld      b,#08*4                         ;b=32 bytes a colorir
transcor1       call    wtvram                          ;loop para enviar
                inc     hl                              ;a cor
                djnz    transcor1                       ;
                ret                                     ;retorna

;a rotina tabnomes preenche um terço da tela com espaços
;em branco usando o caractere 4, já que este caractere
;não foi definido, estando, portanto, em branco. Na verdade,
;todos os caracteres na faixa de 4 a 255 estão em branco,
;pois a rotina transdes só definiu os caracteres de 0 a 3

tabnomes
                ld      hl,(grpnam)                     ;hl aponta para a tab. de
                                                        ;nomes
                ld      b,#00                           ;prepara um loop com 256
                                                        ;repetições
                ld      a,#04                           ;a=núm. do carac. 4 (branco
                                                        ;em branco não definido)
tabnomes1       call    wtvram                          ;preenche um terço da tela
                inc     hl                              ;com espaços em branco
                djnz    tabnomes1                       ;
                ret                                     ;retorna
```

```
rdvram
          ld      a,(versao)      ;obtém a versão do MSX
          or      a               ;é MSX1?
          jr      z,rdvram1       ;Sim, vai para rdvram1
          xor     a               ;Não, inicializa o VDP
          out     (#99),a         ;do MSX2
          ld      a,#8e
          out     (#99),a         ;
rdvram1   ld      a,l             ;informa ao
          out     (#99),a         ;VDP o endereço na
          ld      a,h             ;VRAM onde será
          and     #3f             ;lido o dado
          out     (#99),a         ;
          ex      (sp),hl         ;demora para
          ex      (sp),hl         ;sincronização
          in      a,(#98)         ;lê o dado na VRAM
          ret

wtvram
          push    af              ;salva o dado a ser gravado
          ld      a,(versao)      ;obtém a versão do MSX
          or      a               ;é MSX1?
          jr      z,wtvram1       ;Sim, vai para wtvram1
          xor     a               ;Não, inicializa o VDP
          out     (#99),a         ;do MSX2
          ld      a,#8e
          out     (#99),a
wtvram1   ld      a,l             ;informa ao
          out     (#99),a         ;VDP o endereço na
          ld      a,h             ;VRAM onde o
          and     #3f             ;dado será
          or      #40             ;gravado
          out     (#99),a         ;
          ex      (sp),hl         ;demora para
          ex      (sp),hl         ;sincronização
          pop     af              ;recupera o dado
          out     (#98),a         ;grava o dado
          ret


;Tabela de 32 bytes com os desenhos do radar

desenho1
```

```
                    defb    %00000000
                    defb    %00000100
                    defb    %00001000
                    defb    %00011110
                    defb    %00011110
                    defb    %00011000
                    defb    %00010100
                    defb    %00010000

desenho2
                    defb    %00000000
                    defb    %00111000
                    defb    %01000100
                    defb    %10010010
                    defb    %10010010
                    defb    %01000100
                    defb    %00111000
                    defb    %00010000

desenho3
                    defb    %00000000
                    defb    %01000000
                    defb    %00100000
                    defb    %11110000
                    defb    %11110000
                    defb    %00110000
                    defb    %01010000
                    defb    %00010000

desenho4
                    defb    %00000000
                    defb    %00111000
                    defb    %01111100
                    defb    %11111110
                    defb    %11111110
                    defb    %01111100
                    defb    %00111000
                    defb    %00010000

fim                 equ     $
```

Listagem em linhas DATA do código-objeto do programa para animação em tela gráfica:

```
10 FOR A%=&HD000 TO &HD0CD
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG18.BIN",&HD000,&HD0CD
100 DATA 3A,AF,FC,FE,02,C0,F3,CD,48,D0,CD,59,D0,CD,67,D0
110 DATA 2A,C7,F3,11,70,00,19,AF,CD,8F,D0,5F,7B,3C,E6,03
120 DATA 5F,CD,8F,D0,CD,32,D0,CB,47,28,05,CD,3D,D0,18,EC
130 DATA FB,C9,DB,AA,E6,F0,F6,08,D3,AA,DB,A9,C9,E5,21,00
140 DATA 80,2B,7C,B5,20,FB,E1,C9,2A,CB,F3,11,AD,D0,06,20
150 DATA 1A,CD,8F,D0,13,23,10,F8,C9,2A,C9,F3,3E,F1,06,20
160 DATA CD,8F,D0,23,10,FA,C9,2A,C7,F3,06,00,3E,04,CD,8F
170 DATA D0,23,10,FA,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E
180 DATA D3,99,7D,D3,99,7C,E6,3F,D3,99,E3,E3,DB,98,C9,F5
190 DATA 3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99
200 DATA 7C,E6,3F,F6,40,D3,99,E3,E3,F1,D3,98,C9,00,04,08
210 DATA 1E,1E,18,14,10,00,38,44,92,92,44,38,10,00,40,20
220 DATA F0,F0,30,50,10,00,38,7C,FE,FE,7C,38,10,00
```

Listagem do programa de teste:

```
10 BLOAD"PROG18.BIN"
20 DEFUSR=&HD000
30 SCREEN 2
40 A=USR(0)
```

COMENTÁRIOS SOBRE O PROGRAMA QUE IMPLEMENTA A ANIMAÇÃO DE CARACTERES

Ao executar o programa de teste, você verá a parte de cima de um pequeno radar girando no meio do primeiro terço da tela. Como você mesmo poderá observar, o efeito de animação é provocado pela troca sucessiva do número do desenho na tabela de nomes. Agora que chegamos ao fim da apresentação das rotinas para a parte de vídeo, ficou bem claro que a mais importante de todas as tabelas usadas pelo VDP é a tabela de nomes, por gerenciar todas as demais tabelas. Se aparecerem dúvidas, leia com atenção os comentários feitos nas listagens dos códigos-fonte , tentando, ao mesmo tempo, fazer pequenas modificações para observar os novos resultados. Não tenha medo, pois a linguagem assembly não estraga o seu microcomputador, afinal, esta é a linguagem natural do seu MSX.
Terminada a parte de vídeo, vamos passar agora para o uso de interrupções e hooks no MSX.

BIBLIOGRAFIA RECOMENDADA

INTRODUÇÃO À LINGUAGEM DE MÁQUINA PARA MSX - **Eduardo Alberto Barbosa - Rio de** Janeiro - CIÊNCIA MODERNA COMPUTAÇÃO LTDA.

APROFUNDANDO-SE NO MSX - EDITORA ALEPH

PROGRAMAÇÃO AVANÇADA EM MSX - EDITORA ALEPH

Capítulo 2

OS HOOKS E O INTERPRETADOR BASIC

Neste capítulo, vamos ver um assunto que, apesar de relativamente simples, assusta muitos usuários. Os chamados **hooks** (ganchos) nada mais são do que desvios nas rotinas normais da **BIOS** e do **Interpretador BASIC**. "Como assim?" Ao projetar as rotinas da BIOS e do Interpretador BASIC do MSX, a Microsoft, talvez baseada no sistema criado para o IBM-PC, adotou o sistema de hooks para todas as principais rotinas armazenadas na ROM do MSX. Assim sendo, quando chamamos uma dessas rotinas, como por exemplo a rotina para leitura do teclado, a primeira tarefa realizada pela própria rotina é fazer uma chamada (CALL, em assembly) ao hook correspondente a ela. Como a rotina de inicialização do MSX preenche todos os hooks com o byte #C9 (RET em assembly), a chamada das rotinas aos ganchos geralmente resulta em nada, a menos que o usuário redefina os hooks. Para cada rotina que disponha de um hook existe uma área de 5 bytes reservada nas variáveis do sistema. É nessa área que se situa o hook propriamente dito. Por ser uma área pequena (apenas 5 bytes), o seu uso fica limitado a saltos (JUMPS, em assembly) para rotinas mais complexas criadas pelo usuário. Levando em conta que a tabela de hooks se estende do endereço #FD9A ao #FFC9 e que cada hook gasta 5 bytes, podemos contar, então, com 112 hooks para os propósitos mais variados. Não é o objetivo deste livro apresentar uma tabela com todos os hooks (para tanto, consulte o "*Livro Vermelho do MSX*"), mas ensinar uma utilização mais personalizada do seu computador. Entre os 112 hooks podemos destacar três categorias:

1.Hooks que permitem redefinir rotinas já existentes;

2.Hooks que implementam rotinas não existentes, como por exemplo as rotinas para acesso ao disk drive, e;

3.Hooks que funcionam como interrupções em 60 ciclos.

Neste capítulo, vamos estudar os três tipos acima, começando pelo último.

HOOKS DAS INTERRUPÇÕES

Você já deve ter ouvido falar que o MSX apresenta uma interrupção que "varre" o seu teclado 60 vezes por segundo. O nome interrupção vem do fato de que para o MSX fazer a varredura do teclado, interrompe (suspende) a execução do programa principal (um programa em BASIC ou até em linguagem de máquina) para executar a rotina que faz a varredura. O termo

varredura está ligado à leitura completa do teclado para detectar a tecla ou teclas pressionadas. O tempo em que a execução do programa principal permanece suspensa depende do tamanho e da complexidade da rotina que a interrompeu. Diante deste argumento, você vai concordar que a rotina de interrupção deve ser a mais simples e menor possível, pois, caso contrário, teremos uma execução cada vez mais lenta do programa principal. Para entender como funciona uma interrupção, vejamos o seguinte exemplo: imagine que você esteja escrevendo um livro no seu computador, usando um editor de textos, quando o telefone toca. Como você não sabe se a conversa vai demorar ou não, salva o arquivo editado no disquete antes de atender o telefone. Feito isto, você atende o telefone e deixa o computador ligado, com o cursor na linha do texto em que se encontrava quando o telefone tocou e você salvou o texto no disquete. Podemos então dizer que o computador está à espera de que você termine a interrupção. Assim que você terminar a conversa (interrupção), retorne ao computador e continue o seu trabalho normal. Apesar de simples, o exemplo reflete o que ocorre na **CPU** (unidade de processamento central que, no caso do MSX, é um **Z80**) quando se solicita uma interrupção. Em linhas gerais, as tarefas executadas pelo Z80, no caso de uma interrupção, são as seguintes:

1.Salvar o PC, que vem a ser um registro que aponta para o endereço da instrução sendo executada;

2.Salvar todos os registros;

3.Desviar o PC para o endereço da rotina de interrupção;

4.Executar a rotina de interrupção;

5.Recuperar os registros salvos e o PC, e;

6.Retornar com a execução do programa principal a partir do endereço dado pelo PC.

O PROGRAMA DUMP

Apresentado o mecanismo das interrupções, vamos nos aprofundar um pouco mais no uso de tal facilidade no MSX. Já sabemos que as interrupções no MSX ocorrem a cada 1/60 segundos ou a cada 1,67 centésimos de segundo. Como se vê, é um intervalo de tempo (ou tempo de amostragem, para os mais técnicos) que permite um bom grau de precisão em algumas tarefas, como a de colocar na tela um relógio com segundos. Mas, deixemos o exemplo do relógio para depois, já que se trata de um exemplo clássico que todos conhecem. Que tal algo mais útil para nós, programadores? Há cerca de um ano, vi um programa de **shareware** (programas de demonstração gratuitos) para o IBM-PC que me impressionou pela simplicidade e eficiência. O que esse programa faz é mostrar o conteúdo de diversas posições da memória numa pequena janela na tela, independentemente do programa que o micro esteja executando. Pelas características, fica evidente que se trata de um programa que utiliza os mecanismos de interrupção. Quando o vi em ação, pensei: "Eis um programa que seria útil no MSX". O fato de podermos tirar um raio X da memória durante a execução de programas é fantástico. Suponha, por exemplo, que você deseje acompanhar a evolução do stack pointer

(pilha) durante a execução de uma determinada rotina. Passado o entusiasmo inicial, comecei a me deparar com alguns problemas, entre eles:

1.Como simular o recurso de janelas no MSX?

2.Como imprimir o conteúdo da memória (dump) com a rapidez necessária?

3.Em que endereço da RAM colocar a rotina?

O primeiro problema foi resolvido até de maneira bem simples. Levei em consideração que a depuração (etapa de correção de erros lógicos em programas) se faz usando a tela de texto, e que o MSX permite redefinir o tamanho da tela no que tange ao número de linhas. Diante disto, o primeiro problema desapareceu, pois bastava enganar o MSX a respeito do número de linhas da tela para simular o mecanismo da janela. "Mas, por que é tão importante simular o mecanismo das janelas?" Acontece que, se houver uma mistura de informações na tela, vai ficar extremamente difícil acompanhar tanto os resultados do programa principal como os da interrupção. Para evitar essa confusão, torna-se necessário dividir a tela em duas áreas: uma dedicada exclusivamente ao programa principal e outra dedicada exclusivamente à rotina de interrupção. O interpretador BASIC do MSX faz uso de uma variável do sistema chamada **CRTCNT**, que armazena o número de linhas da tela no modo texto. Ao ligar o MSX, a rotina de inicialização coloca nessa variável o valor 24, que corresponde ao número de linhas numa tela de texto. Entretanto, como a nossa rotina não vai fazer qualquer tipo de chamada às rotinas na ROM do MSX, não existe tal limite para a nossa interrupção. Que podemos, então, fazer? Na época, pensei em reservar um número de linhas que fosse apenas suficiente para os propósitos da interrupção a ser criada. Reservei, então, 4 linhas para a apresentação, por interrupção, de uma determinada região da memória. Para reservar tal número de linhas, basta colocar o valor 20 na variável CRTCNT. Feito isto, a tela de texto passa a apresentar duas divisões: aquela usada pelo interpretador BASIC, que vai da linha 0 à 19, e aquela usada pela rotina de interrupção, que vai da linha 20 à 23. Restava então resolver os 2 últimos problemas. O segundo problema foi resolvido empregando o acesso direto às portas do VDP para a apresentação dos resultados, e a utilização da instrução **DI** (em assembly), que garante a não execução de outras interrupções durante o funcionamento da nossa interrupção. Este último passo é absolutamente necessário, já que não há garantia de que a execução da nossa rotina levará somente 1,67 centésimos de segundo para ser executada (se bem que a nossa rotina certamente gasta muito menos tempo do que esse limite). Se, por um acaso, não fizéssemos isto, poderia ocorrer um pedido de interrupção durante a execução da nossa rotina, que levaria o Z80 a executá-la de novo, acarretando assim um "travamento" do microcomputador. O terceiro problema também foi resolvido facilmente, já que uma rotina de interrupção que se preze não deve ficar à mercê do uso indevido da memória RAM destinada ao BASIC. O interpretador BASIC já deixa livres uns escassos 24Kb de memória RAM; se formos tirar mais ainda para armazenar as nossas rotinas, as coisas podem se complicar. O que me ocorreu, então, foi usar uma pequena parte daqueles 32Kb de RAM que não são usados pelo BASIC. Se você não está familiarizado com o mecanismo de slots do MSX, sugiro que leia algo sobre o assunto no livro ***"Introdução à Linguagem de Máquina para MSX"***, de minha autoria. Como sabemos, esses 32Kb estão localizados nas páginas 0 e 1 do slot que contém 64Kb de memória RAM (slot 2 no Expert e 3 no Hot-Bit). Usando essa memória para armazenar a nossa rotina

de interrupção, chegamos ao milagre de não gastar um byte sequer da memória RAM destinada ao BASIC. Resta-nos agora escolher o hook que será desviado para a nossa rotina. Existem somente dois hooks que podem ser aproveitados para interrupções em 60 ciclos; são eles:

HKEYI usado pela rotina de varredura do teclado

HTIMI usado pelo timer do MSX

O hook HKEYI situa-se no endereço #FD9A, e o hook HTIMI no endereço #FD9F. O único que fica realmente liberado para uso é o primeiro, já que o segundo é usado pela interface de disco para parar e ligar o motor do drive. Resta-nos, portanto, saber como usar os hooks. A própria Microsoft sugere o seguinte uso para os cinco bytes de qualquer hook:

Primeiro byte	:	**RST #30**
Segundo byte	:	Identificação do slot
Terceiro e Quarto bytes	:	Endereço da rotina
Quinto byte	:	RET

A instrução RST #30 nada mais é do que um CALL #0030, só que ocupa apenas um byte, e não três como a instrução CALL. A rotina presente no endereço #0030 da ROM do MSX faz uma chamada interslots, ou seja, ativa a página de memória especificada pelo endereço da rotina no slot desejado, e por fim chama a rotina (CALL). Para tanto, a instrução RST #30 obtém o byte de identificação do slot e o endereço da rotina nos três bytes que a seguem no hook. O nome desta rotina, que se inicia no endereço #0030, é **CALLF**. Como vamos colocar a nossa rotina de interrupção na página 1 (o pedaço de 16Kb que se estende do endereço #4000 ao #7FFF) do slot que contém os 64Kb de RAM, devemos colocar os seguintes dados no hook:

```
RST   #30
DEFB  #02
DEFW  #4000
RET
```

se o micro for um Expert, e:

```
RST   #30
DEFB  #03
DEFW  #4000
RET
```

se o micro for um Hot-Bit. Como o slot da RAM pode variar de máquina para máquina, torna-se necessário fazer uma rotina que descubra o valor exato a colocar no segundo byte do hook. Mais uma vez, sugiro consultar a bibliografia indicada para tirar quaisquer dúvidas sobre o funcionamento dos slots no MSX. Agora que já temos a base necessária, que tal passarmos para as listagens da nossa rotina de interrupção?

Listagem em assembly Z-80 do código-fonte do programa para dump da memória:

```
;programa para fazer dump da memória
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG19.BIN=PROG19.GEN
;
;onde PROG19.GEN é o nome do arquivo-texto com esta
;listagem

versao  equ     #002d
linlen  equ     #f3b0
valtyp  equ     #f663
endini  equ     #f600
tempo   equ     #f602
txtcgp  equ     #f3b7
erafnk  equ     #00cc
crtcnt  equ     #f3b1
hkeyi   equ     #fd9a

                defb    #fe             ;simula em CP/M
                defw    inicio          ;o cabecalho de
                defw    inidump+fimdump-dump+1
                                        ;um arquivo
                defw    inicio          ;.BIN

                org     #9000

inicio
                di                      ;desabilita as interrupções
                in      a,(#a8)         ;lê a config. de slots
                ld      e,a             ;salva em e
                rrca                    ;prepara para ativar
                rrca                    ;a página 1
                or      e               ;
                and     %11111100       ;ativa a página 1 em RAM
                out     (#a8),a         ;
                rrca                    ;descobre o slot da RAM
                rrca                    ;
                and     #03
                ld      (slotnewhk),a   ;guarda o slot
                ld      hl,newhook      ;prepara o desvio
                ld      de,hkeyi        ;do hook
                ld      bc,#0005        ;
                ldir                    ;desvia
                ld      hl,inidump      ;prepara a transferência
```

```
                ld      de,dump           ;da rotina de interrupção
                ld      bc,fimdump-dump;para o end. #4000
                inc     bc
                ldir                      ;transfere
                in      a,(#a8)           ;restabelece a configuração
                and     %11110000         ;original dos slots
                out     (#a8),a           ;
                ld      a,20              ;tira as 4 últimas linhas
                ld      (crtcnt),a;
                ld      a,20              ;inicializa o tempo
                ld      (tempo),a         ;
                ld      hl,#8000          ;inicializa o end. inicial
                ld      (endini),hl       ;
                call    erafnk            ;apaga as teclas de função
                ei                        ;habilita as interrupções
                ret                       ;volta ao BASIC


newhook
                defb    #f7
slotnewhk       defb    #00
                defw    dump
                defb    #c9


inidump equ     $

                org     #4000

dump
                ld      hl,tempo
                ld      a,(temporeal)     ;verifica se já chegou
                inc     a                 ;ao tempo-limite
                cp      (hl)
                ld      (temporeal),a     ;
                ret     nz                ;Não, volta ao interpretador
                xor     a                 ;Sim, zera a variável tempo
                ld      (temporeal),a     ;
                di                        ;desabilita as interrupções
                ld      a,(versao)        ;a=versão do MSX
                or      a                 ;é igual a zero (MSX 1)?
                jr      z,inicio1         ;se for, vai para inicio1
                ld      a,(linlen)        ; o MSX 2 está em 80 colunas?
                cp      41
                jr      c,inicio1         ;Não, pula para inicio1
                ld      a,25              ;a=número de dígitos
                ld      hl,1600           ;hl=20*80=1600=pos. inicial
```

```
            jr      inicio2
inicio1 ld  a,11                ;a=número de dígitos
            ld      hl,800              ;hl=20*40=800=pos. inicial
inicio2 ld  (posini),hl         ;guarda o end. da tab. de
                                ;nomes
            ld      (numdigs),a         ;guarda o número de dígitos
enderdump   ld      de,(endini)         ;obtém o end. de varredura
            ld      hl,(posini)         ;obtém o end. da tab. de nomes
            call    setvdpwt            ;prepara o VDP para escrita
            ld      b,#04               ;b=4 linhas a imprimir
loopdump1   push    bc                  ;salva b
            ld      a,d                 ;a=2 primeiros dígs. do end.
            call    hexa                ;imprime-os
            ld      a,e                 ;a=2 segundos dígs. do end.
            call    hexa                ;imprime-os
            call    espaco              ;imprime um espaço
            ld      a,(numdigs)         ;a=número de dígitos
            ld      b,a                 ;b=número de dígitos
loopdump2   ld      a,(de)              ;obtém o byte no end.
            call    hexa                ;imprime 2 dígitos em hexa
            call    espaco              ;imprime um espaço
            inc     de                  ;incrementa o ponteiro
            djnz    loopdump2           ;continua
            ld      a,(numdigs)         ;teste se impressão em 40
            cp      12                  ;ou em 80 colunas
            jr      nc,loopdump3        ;Se em 80, vai para loopdump3
            ld      a,(de)              ;Se em 40, imprime o último
            call    hexa                ;dígito
loopdump3   inc     de                  ;obtém o próximo endereço
            pop     bc                  ;recupera o cont. de linhas
            djnz    loopdump1           ;continua até terminarem as
                                        ;linhas
            ei                          ;habilita as interrupções
            ret                         ;retorna ao BASIC


;a rotina hexa imprime um valor de 8 bits em hexadecimal

hexa
            push    de                  ;salva o par alterado
            ld      e,a                 ;e=a=valor a imprimir
            and     #f0                 ;obtém os 4 bits mais altos
            rrca                        ;desloca-os para as posições
            rrca                        ;inferiores
            rrca                        ;
            rrca                        ;
            call    hexa1               ;imprime o primeiro dígito
```

```
                ld      a,e             ;recupera o valor
                pop     de              ;recupera o par alterado
                and     #0f             ;obtém os 4 bits inferiores
hexa1   cp      #0a                     ;o dígito é maior ou igual a
                                        ;#A?
                jr      c,hexa2         ;Não, pula para hexa2
                add     a,#07           ;Sim, adiciona #07 ao código
                                        ;em ASCII
hexa2   add     a,#30           ;adiciona #30 (#30=ASCII 0)
                out     (#98),a         ;imprime
                ret                     ;retorna


espaco
                ld      a,#20           ;a=cód. ASCII do espaço
                out     (#98),a         ;imprime
                ret                     ;retorna


setvdpwt
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,wtvram1       ;Sim, vai para wtvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a
wtvram1 ld      a,l                     ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a         ;
                ret                     ;retorna

posini          defw    #0000
numdigs         defb    #00
temporeal       defb    #00

fimdump         equ     $
```

Listagem em linhas DATA do código-objeto programa para dump da memória:

```
10 FOR A%=&H9000 TO &H90EC
20 READ B$
30 POKE A%,VAL("&H"+B$)
```

```
40 NEXT A%
50 BSAVE "PROG19.BIN",&H9000,&H90EC
100 DATA F3,DB,A8,5F,0F,0F,B3,E6,FC,D3,A8,0F,0F,E6,03,32
110 DATA 45,90,21,44,90,11,9A,FD,01,05,00,ED,B0,21,49,90
120 DATA 11,00,40,01,A2,00,03,ED,B0,DB,A8,E6,F0,D3,A8,3E
130 DATA 14,32,B1,F3,3E,14,32,02,F6,21,00,80,22,00,F6,CD
140 DATA CC,00,FB,C9,F7,00,00,40,C9,21,02,F6,3A,A1,40,3C
150 DATA BE,32,A1,40,C0,AF,32,A1,40,F3,3A,2D,00,B7,28,0E
160 DATA 3A,B0,F3,FE,29,38,07,3E,19,21,40,06,18,05,3E,0B
170 DATA 21,20,03,22,9E,40,32,A0,40,ED,5B,00,F6,2A,9E,40
180 DATA CD,86,40,06,04,C5,7A,CD,67,40,7B,CD,67,40,CD,81
190 DATA 40,3A,A0,40,47,1A,CD,67,40,CD,81,40,13,10,F6,3A
200 DATA A0,40,FE,0C,30,04,1A,CD,67,40,13,C1,10,D7,FB,C9
210 DATA D5,5F,E6,F0,0F,0F,0F,0F,CD,76,40,7B,D1,E6,0F,FE
220 DATA 0A,38,02,C6,07,C6,30,D3,98,C9,3E,20,D3,98,C9,3A
230 DATA 2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C
240 DATA E6,3F,F6,40,D3,99,C9,00,00,00,00,00,00
```

Listagem do programa de teste:

```
10 BLOAD"PROG19.BIN",R
20 POKE &HF602,15:REM DEFINE UM NOVO INTERVALO (25 CENTÉSIMOS)
30 POKE &HF600,&H80:POKE &HF601,&H80:REM DEFINE END. INICIAL
40 FOR A%=0 TO 5000
50 NEXT
```

COMENTÁRIOS SOBRE O PROGRAMA DUMP

Se você já tem alguma prática com o assembly e com o mecanismo de slots, não deve ter sentido dificuldade em acompanhar a listagem do código-fonte. Talvez, o fato mais estranho seja a utilização de dois **ORG**s. Esta necessidade se deve ao fato de que o programa PROG19.BIN terá de ser carregado obrigatoriamente na memória RAM do BASIC, pois o comando **BLOAD** só consegue carregar arquivos nessa área (entre os endereços #8000 e #E100, se você estiver usando apenas um drive lógico). Assim sendo, temos de colocar no início uma rotina que faça os ajustes necessários para transferir a nossa rotina para o endereço desejado, além de desviar o hook HKEYI. Coloquei, então, essa rotina a partir da posição #9000 (você pode alterá-la), mas, como a rotina de interrupção será deslocada para o endereço #4000, tive de colocar um segundo ORG para que os deslocamentos relativos à rotina de interrupção se fizessem para os endereços corretos. Este macete vale para todas as ocasiões em que você tenha de carregar um programa numa determinada área e, depois, transferi-lo para a posição correta de execução. Como você vê, não existe grande mistério.

Ao executar o programa de teste, você verá que a linha 20 define um intervalo seguro de 25 centésimos (15/60). Digo **seguro** porque é um intervalo pequeno o suficiente para acompanhar a evolução das modificações feitas em RAM e ao mesmo tempo grande de modo que não trava o micro. Na linha 30, temos a definição do endereço inicial do dump que, no caso, é

#8000. As linhas 40 e 50 são apenas uma pequena demonstração do que a nossa interrupção pode fazer. Você já deve saber que todo o programa em BASIC (como a listagem do programa de teste) termina com uma seqüência de três bytes iguais a #00. Após esses três bytes, começa a área das variáveis inteiras do BASIC, com cada variável ocupando 5 bytes:

Primeiro byte	**:**	**Identificação (#02)**
Segundo e Terceiro bytes	**:**	**Nome da variável**
Quarto e Quinto bytes	**:**	**Valor da variável na forma LSB e MSB**

Ao executar o programa de teste acima, você verá no final da primeira linha (se estiver no modo de 80 colunas do MSX2) ou no final da segunda linha e início da terceira (se você estiver no modo de 40 colunas de qualquer MSX) a seguinte seqüência de bytes escritos na base hexadecimal:

00 00 00 02 41 00 89 13

Se você reparar, durante a execução do programa, os dois últimos bytes acima apresentam uma alteração constante de valores. O valor **89 13** é o que aparece por último quando a execução do programa termina. "Mas, o que vêm a ser esses bytes?" Os três primeiros bytes você já sabe que são o indicador de término do programa em BASIC, o byte **02** é o indicador de variável inteira, o byte **41** é o código em ASCII da letra A, que vem a ser o nome da nossa variável, o byte **00** após o **41** seria o segundo caractere do nome da variável, já que o MSX permite que o nome de cada variável seja composto por até dois caracteres e, por fim, os dois últimos bytes correspondem ao valor da variável A. Dá para perceber que a nossa rotina de interrupção é bastante rápida, já que consegue captar a atualização do valor da variável A feita pelo comando **FOR NEXT** do BASIC. Vamos, então, passar para outro exemplo.

O PROGRAMA PARA ROTAÇÃO POR INTERRUPÇÃO

No início de 1988, elaborei um conjunto de programas utilitários para o MSX-DOS, que recebeu o nome de MSX-DOS Tools Nacional. Uma das rotinas que mais chamava a atenção colocava o meu nome na última linha da tela e fazia tal linha girar para a esquerda num intervalo de tempo constante. Agora, você já sabe que usei uma interrupção para tal efeito. O que vamos ver aqui nada mais é do que uma combinação da rotina acima com a rotina de rotação da tela para a esquerda, apresentada no Capítulo 1. O programa que vamos ver não coloca qualquer mensagem na última linha da tela, de modo que sugiro a você usar um artifício semelhante ao apresentado na listagem do programa de teste.

Listagem em assembly Z-80 do código-fonte do programa para rotar somente a última linha da tela para a esquerda:

```
;programa para rotar a última linha da tela
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG20.BIN=PROG20.GEN
;
;onde PROG20.GEN é o nome do arquivo-texto com esta
;listagem

versao      equ     #002d
linlen      equ     #f3b0
txtnam      equ     #f3b3
valtyp      equ     #f663
argusr      equ     #f7f8
txtcgp      equ     #f3b7
tempo       equ     #f600
erafnk      equ     #00cc
crtcnt      equ     #f3b1
hkeyi       equ     #fd9a

                    defb    #fe                     ;simula em CP/M
                    defw    inicio                  ;o cabecalho de
                    defw    inirota+fimrota-rotaesq+1
                                                    ;um arquivo
                    defw    inicio                  ;.BIN

            org     #9000

inicio
            di                                      ;desabilita as interrupções
            in      a,(#a8)                         ;lê a config. de slots
            ld      e,a                             ;salva em e
            rrca                                    ;prepara para ativar
            rrca                                    ;a página 1
            or      e                               ;
            and     %11111100                       ;ativa a página 1 em RAM
            out     (#a8),a                         ;
            rrca                                    ;descobre o slot da RAM
            rrca                                    ;
            and     #03
            ld      (slotnewhk),a                   ;guarda o slot
            ld      hl,newhook                      ;prepara o desvio
            ld      de,hkeyi                        ;do hook
            ld      bc,#0005                        ;
```

```
            ldir                        ;desvia
            ld      hl,inirota          ;prepara a transferência
            ld      de,rotaesq          ;da rotina de interrupção
            ld      bc,fimrota-rotaesq
            inc     bc
            ldir                        ;transfere
            in      a,(#a8)             ;restabelece a configuração
            and     %11110000           ;original dos slots
            out     (#a8),a             ;
            ld      a,23                ;tira a última linha
            ld      (crtcnt),a;
            ld      a,20                ;inicializa o tempo
            ld      (tempo),a
            call    erafnk              ;apaga as teclas de função
            ei                          ;habilita as interrupções
            ret                         ;volta ao BASIC


newhook
            defb    #f7
slotnewhk   defb    #00
            defw    rotaesq
            defb    #c9


inirota     equu    $

            org     #4000

rotaesq
            ld      a,(temporeal)       ;verifica se o tempo-limite
            ld      hl,tempo            ;foi atingido ou não
            inc     a                   ;
            ld      (temporeal),a       ;
            cp      (hl)                ;
            ret     nz                  ;Não, volta para o BASIC
            di                          ;desabilita as interrupções
            xor     a                   ;zera a variável contadora
            ld      (temporeal),a       ;
            ld      a,(versao)          ;a=versão do MSX
            or      a                   ;é igual a zero (MSX 1)?
            jr      z,rotaesq1          ;Sim, vai para rotaesq1
            ld      a,(linlen)          ;o MSX 2 está em 80 colunas?
            cp      41
            jr      c,rotaesq1          ;Não, pula para rotaesq1
            ld      a,80                ;se estiver a=80 colunas
            jr      rotaesq2            ;pula para rotaesq2 por ser MSX 2
```

```
rotaesq1 ld    a,40                ;a=40 colunas
rotaesq2 ld    (numcol),a          ;coloca o núm. de colunas em
                                   ;numcol
         ld    hl,(txtnam)         ;hl=end. tabela de nomes
         ld    a,(numcol)          ;a=núm. de colunas
         ld    b,23                ;b=núm. da últ. linha
         ld    e,a                 ;de=número de colunas
         ld    d,#00               ;calcula o end. da última
loop1    add   hl,de               ;linha na tabela de nomes
         djnz  loop1               ;
         call  setvdprd            ;prepara o VDP para leitura
         ld    a,(numcol)          ;a=núm. de colunas
         push  hl                  ;salva hl
         ld    hl,bufferlinha      ;hl aponta para o buffer
         ld    b,a                 ;b=núm. de colunas
         call  lelinha             ;lê uma linha
         ld    a,(numcol)          ;a=núm. de colunas
         ld    hl,bufferlinha      ;hl aponta para o buffer
         ld    e,a                 ;de=núm. de colunas
         ld    d,#00               ;
         ld    a,(hl)              ;a=primeiro carac. da linha
         add   hl,de               ;hl=aponta para o fim da
                                   ;linha+1
         ld    (hl),a              ;coloca o prim. carac. na últ.
                                   ;pos.
         pop   hl                  ;recupera hl
         call  setvdpwt            ;prepara o VDP para escrita
         ld    hl,bufferlinha+1    ;hl aponta para o buffer+1
         ld    a,(numcol)          ;a=núm. de colunas
         ld    b,a                 ;b=núm. de colunas
         call  esclinha            ;envia a linha já rotada
         ei                        ;habilita as interrupções
         ret                       ;retorna ao BASIC

lelinha
         in    a,(#98)             ;lê o carac. da VRAM
         ld    (hl),a              ;salva-o no buffer
         inc   hl                  ;incrementa o ponteiro
         djnz  lelinha             ;prepara a próxima leitura
         ret                       ;retorna

esclinha
         ld    a,(hl)              ;lê o carac. do buffer
         out   (#98),a             ;escreve-o na VRAM
         inc   hl                  ;incrementa o ponteiro
```

```
                djnz    esclinha        ;prepara a próxima escrita
                ret                     ;retorna

setvdprd
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,rdvram1       ;Sim, vai para rdvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a         ;
rdvram1 ld      a,l                     ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde será
                and     #3f             ;lido o dado
                out     (#99),a         ;
                ret                     ;retorna

setvdpwt
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,wtvram1       ;Sim, vai para wtvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a
wtvram1 ld      a,l                     ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a         ;
                ret                     ;retorna


temporeal       defb    #00
bufferlinha     defs    81
numcol          defb    #00

fimrota         equ     $
```

Listagem em linhas DATA do código-objeto do programa para rotar somente a última linha da tela para a esquerda:

```
10 FOR A%=&H9000 TO &H912F
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG20.BIN",&H9000,&H912F
100 DATA F3,DB,A8,5F,0F,0F,B3,E6,FC,D3,A8,0F,0F,E6,03,32
110 DATA 3C,90,21,3B,90,11,9A,FD,01,05,00,ED,B0,21,40,90
120 DATA 11,00,40,01,EE,00,03,ED,B0,DB,A8,E6,F0,D3,A8,3E
130 DATA 17,32,B1,F3,3E,14,32,00,F6,FB,C9,F7,00,00,40,C9
140 DATA 3A,9B,40,21,00,F6,3C,32,9B,40,BE,C0,F3,AF,32,9B
150 DATA 40,3A,2D,00,B7,28,0B,3A,B0,F3,FE,29,38,04,3E,50
160 DATA 18,02,3E,28,32,ED,40,2A,B3,F3,3A,ED,40,06,17,5F
170 DATA 16,00,19,10,FD,CD,6D,40,3A,ED,40,E5,21,9C,40,47
180 DATA CD,5F,40,3A,ED,40,21,9C,40,5F,16,00,7E,19,77,E1
190 DATA CD,83,40,21,9D,40,3A,ED,40,47,CD,66,40,FB,C9,DB
200 DATA 98,77,23,10,FA,C9,7E,D3,98,23,10,FA,C9,3A,2D,00
210 DATA B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F
220 DATA D3,99,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99
230 DATA 7D,D3,99,7C,E6,3F,F6,40,D3,99,C9,00,00,00,00,00
240 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
250 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
260 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
270 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
280 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
290 DATA 00,00,00
```

Listagem do programa de teste:

```
10 KEYOFF
20 IF (PEEK (&HF3B0) >40) THEN CL%=80 ELSE CL%=40
30 A$="Programa de demonstração"
40 FOR A%=0 TO LEN(A$)-1
50 VPOKE (23*CL%+A%),ASC(MID$(A$,A%+1,1))
60 NEXT A%
70 BLOAD"PROG20.BIN",R
```

COMENTÁRIOS SOBRE
O PROGRAMA PARA ROTAÇÃO POR INTERRUPÇÃO

Se, por um acaso, você não gostou da velocidade de rotação da última linha, tente alterar o valor em #F600. Antes de fazer tal alteração, convém ficar ciente de que valores abaixo de

8 podem congelar o seu micro,devido à "lentidão", não da nossa rotina, mas da rotina **RST #30** que usamos para desviar o hook. No meu ponto de vista, o comando abaixo imprimirá uma velocidade bem razoável:

POKE &HF600,10

Entre com o comando acima após o interpretador BASIC exibir a mensagem **Ok**, quando terminar a execução do programa de teste. Infelizmente, temos um problema: não podemos usar o comando **CLS** do BASIC para apagar a tela, já que este comando ignora o valor colocado em **CRTCNT**. Não adianta enganar o interpretador BASIC no que diz respeito ao número de linhas na tela, no caso do comando CLS. Observe que o artifício de se colocar um valor menor que 24 na variável **CRTCNT** é válido para a exibição das teclas de funções e para os comandos de edição (seta para baixo, seta para a esquerda, etc), embora não o seja para o comando CLS. Como você pode perceber, temos aí um bug (situação de erro não prevista) no projeto do MSX. Ainda neste capítulo, vamos aprender como desviar o interpretador de erros (através do respectivo hook) para implementarmos novos comandos no interpretador BASIC. Nesse momento, vamos criar o comando **CLRSCR**, que apagará a tela de acordo com um determinado argumento, respeitando inclusive o número de linhas colocado em **CRTCNT**. Vamos ver, agora, um efeito igualmente interessante.

O PROGRAMA PARA ANIMAÇÃO GRÁFICA POR INTERRUPÇÃO

No primeiro capítulo, cheguei a mencionar que se poderia obter efeitos de animação interessantes usando-se as interrupções. Talvez você fique surpreso ao saber que os jogos da famosa Konami são controlados por uma interrupção, ou seja, a parte do programa que lê os joysticks/teclado e toma as ações cabíveis é resultado de um desvio do hook **HKEYI**. Isto tem certa lógica, pois, como sabemos, as colisões de sprites (leia-se colisão de aviões com mísseis, etc) são testadas a cada ciclo de 60 Hz, exatamente na mesma freqüência em que são ativadas as rotinas de interrupção. Se levarmos em conta que a interrupção de 60 Hz é gerada pelo VDP (processador de vídeo), temos então um perfeito sincronismo entre o teste da colisão de sprites e o início da execução das rotinas de interrupção. O exemplo que vamos ver nada mais é do que a conjunção de dois exemplos do Capítulo 1: o programa para movimentação de sprites e o programa para animação de um pequeno radar. Neste caso, a rotina para a qual será desviado o hook será a de animação do radar. Você vai perceber que se consegue um efeito bem interessante. Vamos, então, às listagens:

Listagem em assembly Z-80 do código-fonte do programa para fazer animação por interrupção:

```
;programa para movimentar sprites pela tela
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG21.BIN=PROG21.GEN
;
;onde PROG21.GEN é o nome do arquivo-texto com esta
;listagem

versao      equ     #002d
kilbuf      equ     #0156
grpnam      equ     #f3c7
grpcol      equ     #f3c9
grpcgp      equ     #f3cb
grppat      equ     #f3cf
grpatr      equ     #f3cd
scrmod      equ     #fcaf
hkeyi       equ     #fd9a

            defb    #fe                 ;simula em CP/M
            defw    inicio              ;o cabecalho de
            defw    fim                 ;um arquivo
            defw    inicio              ;.BIN

            org     #d000

inicio
            ld      a,(scrmod)          ;o MSX está no modo
            cp      #02                 ;gráfico?
            ret     nz                  ;Não, retorna ao BASIC
            ld      hl,(grppat)         ;hl aponta para a tab. de
                                        ;padrões
            ld      de,desenho          ;de aponta para o
                                        ;desenho
            ld      b,#08               ;b=8 bytes que formam o
                                        ;desenho
loop1       ld      a,(de)              ;este loop envia o
                                        ;desenho
            call    wtvram              ;para a tabela de
                                        ;padrões
            inc     de                  ;dos sprites na RAM de
                                        ;vídeo
            inc     hl                  ;(VRAM)
            djnz    loop1               ;
```

```
        ld      hl,(grpatr)      ;hl aponta para a tab.
                                 ;de atributos
        ld      de,atributos     ;de aponta para os
                                 ;atributos
        ld      b,#04            ;b=4=núm. de bytes dos
                                 ;atributos
loop2   ld      a,(de)           ;este loop envia os
                                 ;atributos
        call    wtvram           ;para a tabela de
                                 ;atributos
        inc     de               ;dos sprites na RAM de
                                 ;vídeo
        inc     hl               ;(VRAM)
        djnz    loop2            ;
        call    transdes         ;chama a rotina transdes
        call    transcor         ;chama a rotina transcor
        call    tabnomes         ;chama a rotina tabnomes
        di                       ;desabilita as
                                 ;interrupções
        ld      hl,(grpnam)      ;hl aponta para a tab.
                                 ;de nomes
        ld      de,32*3+16       ;de=>Y=3 X=16
        add     hl,de            ;hl aponta para a
                                 ;posição (X,Y)
        ld      (posicao),hl     ;salva a posição de
                                 ;escrita
        ld      a,15             ;temporiza em 0.25
                                 ;segundo
        ld      (tempolim),a     ;
        xor     a                ;guarda o caractere
                                 ;atual
        ld      (desatual),a     ;
        ld      (tempo),a        ;zera a variável tempo
        ld      hl,novohook      ;prepara o desvio do
                                 ;hook
        ld      de,hkeyi         ;
        ld      bc,#0005
        ldir
        ei                       ;habilita as
                                 ;interrupções

loopprin
        call    letecla          ;lê as teclas do cursor
                                 ;e a barra de espaço
        push    af               ;salva o valor lido
        bit     7,a              ;a seta para direita foi
                                 ;pressionada?
```

```
call    z,direita           ;Sim, chama a rotina
                            ;direita
pop     af                  ;recupera o valor lido
push    af                  ;torna a salvar
bit     4,a                 ;a seta para esquerda
                            ;foi pressionada?
call    z,esquerda          ;Sim, chama a rotina
                            ;esquerda
pop     af                  ;recupera o valor lido
push    af                  ;torna a salvar
bit     5,a                 ;a seta para cima foi
                            ;pressionada?
call    z,cima              ;Sim, chama a rotina
                            ;cima
pop     af                  ;recupera o valor lido
push    af                  ;torna a salvar
bit     6,a                 ;a seta para baixo foi
                            ;pressionada?
call    z,baixo             ;Sim, chama a rotina
                            ;baixo
pop     af                  ;recupera o valor lido
bit     0,a                 ;a barra de espaço foi
                            ;pressionada?
jr      z,basic             ;Sim, prepara a volta ao
                            ;BASIC
call    espera              ;Não, espera para
                            ;diminuir a
                            ;velocidade do movimento
jr      loopprin            ;fecha o loop


di                          ;desabilita as
                            ;interrupções
ld      hl,hkeyi            ;desfaz o desvio do
                            ;hook.
ld      de,hkeyi+#01        ;Etapa necessária porque
ld      (hl),#c9            ;o MSX retorna para o
ld      bc,#0004            ;modo texto assim que
                            ;terminar
ldir                        ;o programa em BASIC.
ei                          ;habilita as
                            ;interrupções
call    kilbuf              ;limpa o buffer do
                            ;teclado
ret                         ;retorna ao BASIC
```

```
;a rotina radar é a responsável pela animação do radar
;usando o desvio da interrupção HKEYI

radar
        ld      hl,tempolim       ;hl=>variável tempo-
                                  ;limite
        ld      a,(tempo)         ;a=tempo atual
        inc     a                 ;incrementa tempo atual
        ld      (tempo),a         ;guarda o novo valor
        cp      (hl)              ;tempo atual=tempo-
                                  ;limite?
        ret     nz                ;Não, volta sem animação
        di                        ;Sim, desabil. as
                                  ;interrupções
        xor     a                 ;zera a var. tempo atual
        ld      (tempo),a         ;
        ld      hl,(posicao)      ;obtém a pos. de escrita
        ld      a,(desatual)      ;obtém o núm. do desenho
        and     #03               ;garante estar entre 0 e
                                  ;3
        call    wtvram            ;escreve o caractere
        inc     a                 ;incrementa o núm do
                                  ;caractere
        ld      (desatual),a      ;guarda para a próx.
                                  ;interrupção
        ei                        ;habilita as
                                  ;interrupções
        ret                       ;retorna


novohook
        defb    #c3               ;JUMP
        defw    radar             ;chama a rotina radar
        defb    #c9               ;RET
        defb    #c9               ;RET


letecla
        in      a,(#aa)           ;prepara o ppi para ler
        and     #f0               ;a linha (#08) com as
        or      #08               ;informações sobre as
        out     (#aa),a           ;teclas do cursor
        in      a,(#a9)           ;lê a coluna selecionada
        ret                       ;volta
```

```
direita
          ld      hl,(grpatr)          ;hl aponta para o
                                       ;atributo X
                                       ;na VRAM
          inc     hl
          call    rdvram               ;lê a posição x do
                                       ;sprite
          inc     a                    ;incrementa-a
          call    wtvram               ;informa ao VDP a nova
                                       ;posição
          ret                          ;retorna

esquerda
          ld      hl,(grpatr)          ;hl aponta para o
                                       ;atributo X
                                       ;na VRAM
          inc     hl
          call    rdvram               ;lê a posição x do
                                       ;sprite
          dec     a                    ;decrementa-a
          call    wtvram               ;informa ao VDP a nova
                                       ;posição
          ret                          ;retorna

cima
          ld      hl,(grpatr)          ;hl aponta para o
                                       ;atributo Y
                                       ;na VRAM
          call    rdvram               ;lê a posição y do
                                       ;sprite
          or      a                    ;é zero?
          jr      z,cima1              ;Sim, vai para cima1
          dec     a                    ;Não, decrementa y (sobe
                                       ;o sprite)
          jr      cima2                ;vai para cima2
cima1     ld      a,191                ;coloca o sprite na
                                       ;última linha
cima2     call    wtvram               ;informa ao VDP a nova
                                       ;posiçao
          ret                          ;retorna

baixo
          ld      hl,(grpatr)          ;hl aponta para o
                                       ;atributo Y
```

```
                                        ;na VRAM
            call     rdvram             ;lê a posição y do
                                        ;sprite
            cp       191                ;já está na última
                                        ;linha?
            jr       z,baixo1           ;Sim, vai para baixo1
            inc      a                  ;Não, incrementa y
                                        ;(desce o sprite)
            jr       baixo2             ;vai para baixo2
baixo1      xor      a                  ;a=0=primeira linha
baixo2      call     wtvram             ;informa ao VDP a nova
                                        ;posição
            ret                         ;retorna


espera
            ld       hl,#1000           ;provoca um retardo
loopesp     dec      hl                 ;baseado em sucessivos
            ld       a,h                ;decrementos no valor de
            or       l                  ;hl
            jr       nz,loopesp         ;
            ret                         ;


;a rotina transdes transfere os desenhos do radar para a
;tabela de padrões

transdes
            ld       hl,(grpcgp)        ;hl aponta para a tab.
                                        ;de padrões
            ld       de,desenho1        ;de aponta para o desenho1
            ld       b,#08*4            ;b=8 bytes por desenho*4
                                        ;desenhos
transdes1   ld       a,(de)             ;este loop transfere
            call     wtvram             ;todos os
            inc      de                 ;desenhos para a RAM
            inc      hl                 ;de vídeo
            djnz     transdes1          ;
            ret                         ;retorna


;a rotina transcor fornece a cor dos 4 desenhos
;(caracteres) transferidos para a tabela de padrões

transcor
```

```
               ld      hl,(grpcol)        ;hl aponta para a tabela
                                          ;das cores
               ld      a,#f1              ;branco (f=15) para a cor
                                          ;de frente e
                                          ;preto (1) para a cor de
                                          ;fundo
               ld      b,#08*4            ;b=32 bytes a colorir
transcor1      call    wtvram             ;loop para enviar
               inc     hl                 ;a cor
               djnz    transcor1          ;
               ret                        ;retorna


;a rotina tabnomes limpa as 4 primeiras posições da tabela
;de nomes para que os 4 desenhos do radar não apareçam

tabnomes
               ld      hl,(grpnam)        ;hl aponta para a tab. de
                                          ;nomes
               ld      a,#04              ;a=núm. do carac. em
                                          ;branco
               ld      b,a
tabnomes1      call    wtvram             ;limpa as 4 primeiras
               inc     hl                 ;posições da tab. de
                                          ;nomes
               djnz    tabnomes1
               ret                        ;retorna


rdvram
               di                         ;desabilita as
                                          ;interrupções
               ld      a,(versao)         ;obtém a versão do MSX
               or      a                  ;é MSX1?
               jr      z,rdvram1          ;Sim, vai para rdvram1
               xor     a                  ;Não, inicializa o VDP
               out     (#99),a            ;do MSX2
               ld      a,#8e
               out     (#99),a            ;
rdvram1        ld      a,l                ;informa ao
               out     (#99),a            ;VDP o endereço na
               ld      a,h                ;VRAM onde será
               and     #3f                ;lido o dado
               out     (#99),a            ;
               ex      (sp),hl            ;demora para
```

```
            ex      (sp),hl            ;sincronização
            in      a,(#98)            ;lê o dado na VRAM
            ei                         ;habilita as
                                       ;interrupções
            ret

wtvram
            di                         ;desabilita as
                                       ;interrupções
            push    af                 ;salva o dado a ser
                                       ;gravado
            ld      a,(versao)         ;obtém a versão do MSX
            or      a                  ;é MSX1?
            jr      z,wtvram1          ;Sim, vai para wtvram1
            xor     a                  ;Não, inicializa o VDP
            out     (#99),a            ;do MSX2
            ld      a,#8e
            out     (#99),a
wtvram1     ld      a,l                ;informa ao
            out     (#99),a            ;VDP o endereço na
            ld      a,h                ;VRAM onde o
            and     #3f                ;dado será
            or      #40                ;gravado
            out     (#99),a            ;
            ex      (sp),hl            ;demora para
            ex      (sp),hl            ;sincronização
            pop     af                 ;recupera o dado
            out     (#98),a            ;grava o dado
            ei                         ;habilita as
                                       ;interrupções
            ret


;tabela de 8 bytes com o desenho do boneco

desenho
            defb    %00111000
            defb    %00101000
            defb    %00111000
            defb    %00010000
            defb    %00111000
            defb    %01010100
            defb    %00101000
            defb    %01000100
```

```
;tabela dos atributos do sprite

atributos
y               defb    86                  ;coordenadas do ponto
x               defb    128                 ;central da tela
modelo          defb    0                   ;primeiro plano
cor             defb    15                  ;cor branca para o
                                            ;sprite

;tabela de 32 bytes com os desenhos do radar

desenho1
                defb    %00000000
                defb    %00000100
                defb    %00001000
                defb    %00011110
                defb    %00011110
                defb    %00011000
                defb    %00010100
                defb    %00010000

desenho2
                defb    %00000000
                defb    %00111000
                defb    %01000100
                defb    %10010010
                defb    %10010010
                defb    %01000100
                defb    %00111000
                defb    %00010000

desenho3
                defb    %00000000
                defb    %01000000
                defb    %00100000
                defb    %11110000
                defb    %11110000
                defb    %00110000
                defb    %01010000
                defb    %00010000

desenho4
                defb    %00000000
                defb    %00111000
                defb    %01111100
                defb    %11111110
```

```
                defb    %11111110
                defb    %01111100
                defb    %00111000
                defb    %00010000


posicao         defw    #00
desatual        defb    #00
tempolim        defb    #00
tempo           defb    #00

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa para fazer animação por interrupção:

```
10 FOR A%=&HD000 TO &HD19D
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG21.BIN",&HD000,&HD19D
100 DATA 3A,AF,FC,FE,02,C0,2A,CF,F3,11,6C,D1,06,08,1A,CD
110 DATA 4C,D1,13,23,10,F8,2A,CD,F3,11,74,D1,06,04,1A,CD
120 DATA 4C,D1,13,23,10,F8,CD,04,D1,CD,15,D1,CD,23,D1,F3
130 DATA 2A,C7,F3,11,70,00,19,22,98,D1,3E,0F,32,9B,D1,AF
140 DATA 32,9A,D1,32,9C,D1,21,AF,D0,11,9A,FD,01,05,00,ED
150 DATA B0,FB,CD,B4,D0,F5,CB,7F,CC,BF,D0,F1,F5,CB,67,CC
160 DATA CB,D0,F1,F5,CB,6F,CC,D7,D0,F1,F5,CB,77,CC,E9,D0
170 DATA F1,CB,47,28,05,CD,FB,D0,18,D8,F3,21,9A,FD,11,9B
180 DATA FD,36,C9,01,04,00,ED,B0,FB,CD,56,01,C9,21,9B,D1
190 DATA 3A,9C,D1,3C,32,9C,D1,BE,C0,F3,AF,32,9C,D1,2A,98
200 DATA D1,3A,9A,D1,E6,03,CD,4C,D1,3C,32,9A,D1,FB,C9,C3
210 DATA 8D,D0,C9,C9,DB,AA,E6,F0,F6,08,D3,AA,DB,A9,C9,2A
220 DATA CD,F3,23,CD,30,D1,3C,CD,4C,D1,C9,2A,CD,F3,23,CD
230 DATA 30,D1,3D,CD,4C,D1,C9,2A,CD,F3,CD,30,D1,B7,28,03
240 DATA 3D,18,02,3E,BF,CD,4C,D1,C9,2A,CD,F3,CD,30,D1,FE
250 DATA BF,28,03,3C,18,01,AF,CD,4C,D1,C9,21,00,10,2B,7C
260 DATA B5,20,FB,C9,2A,CB,F3,11,78,D1,06,20,1A,CD,4C,D1
270 DATA 13,23,10,F8,C9,2A,C9,F3,3E,F1,06,20,CD,4C,D1,23
280 DATA 10,FA,C9,2A,C7,F3,3E,04,47,CD,4C,D1,23,10,FA,C9
290 DATA F3,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3
300 DATA 99,7C,E6,3F,D3,99,E3,E3,DB,98,FB,C9,F3,F5,3A,2D
310 DATA 00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6
320 DATA 3F,F6,40,D3,99,E3,E3,F1,D3,98,FB,C9,38,28,38,10
330 DATA 38,54,28,44,56,80,00,0F,00,04,08,1E,1E,18,14,10
```

```
340 DATA 00,38,44,92,92,44,38,10,00,40,20,F0,F0,30,50,10
350 DATA 00,38,7C,FE,FE,7C,38,10,00,00,00,00,00,00
```

Listagem do programa de teste:

```
100 BLOAD"PROG21.BIN"
110 SCREEN 2
120 LINE (10,10)-(40,40),15,BF
130 LINE (215,150)-(245,180),12,BF
140 LINE (215,10)-(245,40),13,BF
150 LINE (10,150)-(40,180),14,BF
160 DEFUSR=&HD000
170 A=USR(0)
```

COMENTÁRIOS SOBRE O PROGRAMA PARA ANIMAÇÃO GRÁFICA POR INTERRUPÇÃO

Embora o programa acima seja basicamente o resultado da união de dois programas do Capítulo 1, temos aqui algumas diferenças:

1.A rotina para animação do radar já não é feita pelo programa em si, mas sim por uma rotina de interrupção.

2.As rotinas de acesso ao vídeo e para leitura direta da linha do teclado, que fornece o status das teclas de movimentação do cursor e da barra de espaço, não desativam as interrupções de forma permanente. Isto é absolutamente necessário para que haja uma animação do radar, pois se as interrupções forem permanentemente desabilitadas (DI em assembly), a rotina de interrupção que faz a animação do radar não será ativada.

3.O hook da interrupção foi desviado usando-se a instrução **JUMP**, e não a **RST #30**, como recomenda a Microsoft. Neste ponto, houve apenas a escolha de uma alternativa mais simples e rápida. Acontece que a rotina ativada pela instrução **RST #30** é destinada basicamente à chamada de rotinas em slots/páginas diferentes, o que evidentemente não é o nosso caso, pois a nossa rotina ocupa a memória RAM destinada ao BASIC, não havendo, portanto, qualquer necessidade de chamadas interslots. Se a nossa rotina se iniciasse, digamos, no endereço #4000, haveria a necessidade de usar a instrução RST #30 para ativar a nossa interrupção (como usamos no primeiro e segundo exemplos), já que o endereço #4000 está fora da RAM destinada ao BASIC.

4.Neste exemplo, o desvio do hook é destruído na volta ao BASIC. Antes de comentar a forma como se "processa" um desvio no hook, gostaria de comentar o motivo de tal destruição. Ocorre que, ao voltar para o modo de comando (o modo em que você entra com as linhas de um programa), o MSX se coloca automaticamente no modo texto. Desta forma, não seria inteligente deixar ativo um hook que foi projetado para funcionar no modo gráfico. A presença

de tal hook no modo texto só serviria para diminuir a velocidade de execução do interpretador BASIC, além de alterar continuamente a tabela de padrões de um MSX 2 no modo de 80 colunas. Para se destruir um hook, basta preencher o seu primeiro byte com o valor **#C9** (instrução RET em assembly). No nosso caso, resolvi preencher todos os cinco bytes com esse valor, usando a instrução **LDIR**. Agora, um detalhe importantíssimo: **NUNCA ALTERE UM HOOK DE INTERRUPÇÃO (OS DOIS PRIMEIROS HOOKS) COM AS INTERRUPÇÕES HABILITADAS**; dê sempre um comando DI antes de alterar um destes dois hooks.

5.Na volta ao BASIC, o programa acima, além de destruir o hook, faz uma chamada à rotina **KILBUF**, para promover uma limpeza do buffer do teclado. Nos programas do Capítulo 1, onde não existiam interrupções, não houve necessidade de tal chamada. Acontece que agora habilitamos as interrupções para poder animar o radar. Como conseqüência, habilitamos também as interrupções para leitura do teclado e para controle do motor do(s) drive(s). Já que o sprite é movimentado pelas teclas do cursor, se não promovermos uma limpeza do buffer do teclado na volta ao BASIC, todos esses movimentos se transportarão para o modo de comando, deslocando o cursor para posições imprevisíveis.

Esgotado o assunto das interrupções, vamos para um estudo mais detalhado da interpretação dos erros no BASIC do MSX.

O INTERPRETADOR BASIC E OS ERROS

Você já deve ter notado que se entrar com um comando maluco, como por exemplo **MSX**, o interpretador BASIC do seu MSX exibirá a seguinte mensagem:

ERRO DE SINTAXEse for um Hot-Bit
Syntax errorse for um Expert

Se conseguíssemos interceptar a rotina da mensagem de erro e promover uma pesquisa sobre o tipo de erro, poderíamos levar alguma vantagem, não? "Mas, que tipo de vantagem?" Imagine que você tenha entrado com o seguinte comando:

CLRSCR 1

Como sabemos, tal comando não existe no MSX. Se você entrasse com o comando acima, o interpretador iria exibir de imediato a mensagem de erro de sintaxe. Por outro lado, se você conseguisse desviar a rotina de erro para uma rotina que analisasse o tipo de erro e a sentença que o causou, poderia facilmente implementar novos comandos ao BASIC. Tal "macete" é possível em todos os microcomputadores que usem o BASIC da Microsoft, como o MSX. Para entender como tudo isto funciona, precisamos de alguns princípios básicos.

ARMAZENAMENTO DE UMA LINHA EM BASIC

Para poder entender como o interpretador BASIC funciona, temos de entender o esquema de armazenamento de uma linha em BASIC na memória RAM do MSX. Quando entramos com uma linha em BASIC, tal como:

10 FOR A%=1 TO 5000

o interpretador BASIC transforma os caracteres em ASCII da linha acima em códigos próprios, de modo a otimizar o espaço gasto pela linha. Desta forma, apesar da linha ter sido entrada com 19 caracteres ASCII de comprimento, ocupará, na verdade, 18 posições de memória. Antes de mais nada, vamos aproveitar o programa DUMP (programa 19) para nos ajudar nesse entendimento. Siga os seguintes passos:

1.Vá para o BASIC.

2.Carregue o programa 19 com o seguinte comando:

BLOAD"PROG19.BIN",R

3.Digite **NEW** e aperte Enter/Return para apagar qualquer programa em BASIC que, por ventura, esteja na memória.

4.Entre com a linha acima (10 FOR A%=1 TO 5000).

Ao terminar de digitar e dar entrada à linha acima, você verá que as 2 primeiras linhas das 4 produzidas pelo programa DUMP (programa 19) serão as seguintes (assumindo que você esteja no modo de 40 colunas):

8000 00 12 80 0A 00 82 20 41 25 EF 12 20
800C D9 20 1C 88 13 00 00 00 00 00 00 00

O que significam todos estes números em hexadecimal? O valor contido em #8000, #00, indica a presença ou não de um erro incontornável. Para entender como ele funciona, faça o seguinte:

1.Digite **RUN** e aperte Enter/Return. Após este comando, o MSX responderá com a mensagem **Ok**. Embora você não tenha entrado com mais nenhuma linha, a linha acima não possui qualquer erro de sintaxe. Ao receber o comando RUN, o interpretador BASIC executa o comando **FOR** contido na linha de exemplo. Ao executar o comando FOR, o interpretador acaba por inicializar a variável A% com o valor inicial 1, e retorna para o modo de comando com a mensagem Ok, por não ter encontrado o **NEXT** correspondente ao comando FOR.

2.Após o MSX ter exibido a mensagem **Ok**, entre com a seguinte linha no modo direto:

POKE &H8000,255

3.Digite novamente **RUN** e aperte a tecla Enter/Return. O que aconteceu? O seu MSX respondeu com uma mensagem de erro, ao invés da mensagem **Ok**. Para que o seu MSX volte ao normal, entre com a seguinte linha no modo direto:

POKE &H8000,0

Como você pode ver, a posição de memória #8000 é usada para sinalizar o acontecimento de um erro inesperado. Passemos então para a posição #8001. Os endereços #8001 e #8002 armazenam o endereço da próxima linha em BASIC, que no nosso caso é #8012 (não se esqueça que o Z80 trabalha na forma LSB e MSB, no caso, #12 #80), como você pode verificar na listagem produzida pelo programa DUMP. Os endereços #8003 e #8004 contêm o número da linha (10=#0A) também na forma LSB e MSB. Como você poderá constatar, o valor contido nestas duas posições é #000A=10. O endereço #8005 contém o valor #82, que nada mais é do que o "token" do comando FOR. Como já tinha afirmado, o interpretador BASIC traduz a linha recém-entrada para códigos próprios, visando a otimização do espaço ocupado pela própria linha. Estes códigos recebem o nome genérico de **token**. Você pode perceber a vantagem de tal método olhando para o exemplo do comando FOR, que na sua forma literal gasta 3 caracteres ou 3 bytes, e na sua forma "tokenizada" gasta apenas 1 byte. O endereço #8006 armazena o valor #20, que nada mais é do que o código ASCII do caractere de espaço em branco entre o comando FOR e A%. O endereço #8007 contém o valor #41, que vem a ser o código ASCII da letra A. O endereço #8008 contém o valor #25, que nada mais é do que o código ASCII do caractere %. O endereço #8009 contém, então, o token do sinal de igual, no caso #EF. O endereço #800A contém o token do número 1, no caso #12 (consulte o livro "***PROGRAMAÇÃO AVANÇADA EM MSX***", para maiores detalhes). O endereço #800B, por sua vez, contém o código ASCII do caractere de espaço entre 1 e TO. O endereço #800C contém o token do comando TO. O endereço #800D armazena o código ASCII do caractere de espaço entre TO e 5000. O endereço #800E contém um valor de identificação que avisa que o número contido nos dois endereços de memória seguintes é um inteiro na faixa de 256 a 32767. Os endereços #800F e #8010 contêm, respectivamente, o valor LSB (#88) e MSB (#13) do número inteiro. Vejamos se isto está correto:

#13=19 em decimal
#88=136 em decimal
Número final=19*256+136=5000

Como se percebe pelos cálculos acima, o valor armazenado na forma LSB e MSB está correto. O endereço #8011 contém o valor #00 que, por sua vez, representa o papel de indicador de fim de linha. Os endereços #8012 e #8013, que correspondem ao início de uma nova linha (lembre-se do conteúdo em #8001 e #8002), apresentam o valor #0000. Este fato indica que a linha anterior foi a última do programa em BASIC. A partir desta constatação, podemos concluir que um programa em BASIC termina com uma seqüência de três endereços de memória contendo o valor #00. A partir do endereço #8014, temos a área da memória RAM destinada ao armazenamento das variáveis (inteiras, strings, arrays, etc). Você pode conferir a validade de tal endereço com o seguinte comando:
PRINT HEX$(PEEK(&HF6C2)+256*PEEK(&HF6C3))

Ao entrar com o comando acima, o seu MSX responderá com 8014, que, como já sabemos, corresponde ao início da área destinada ao armazenamento das variáveis do programa em BASIC. "Mas, o que significam os endereços #F6C2 e #F6C3?" Esses endereços estão na área das variáveis do sistema e são atualizados continuamente pelo interpretador BASIC à medida em que vamos retirando ou acrescentando linhas a um programa.

ARMAZENAMENTO DE UMA VARIÁVEL INTEIRA

Com a linha de exemplo anterior (10 FOR A%=1 TO 5000) e com o programa DUMP funcionando, digite o comando RUN e pressione a tecla Enter/Return. Você vai observar que a segunda linha produzida pelo programa DUMP apresenta os seguintes dados:

800C D9 20 1C 88 13 00 00 00 02 41 00 01

O que temos agora? A resposta é bem simples: já sabemos que a partir do endereço #8014 (somente neste exemplo) se inicia a área das variáveis, de modo que o valor #02 em #8014 é um byte de identificação usado para sinalizar uma variável inteira (#03 para string, etc); o valor #41 em #8015 é o código ASCII da primeira letra da variável (A, neste caso); o valor #00 em #8016 seria a segunda letra da variável (nenhuma=nul=#00, neste caso), já que o interpretador BASIC do MSX permite até dois caracteres no nome de uma variável (por isso, os nomes ED e EDU se referem à mesma variável no MSX); o valor #01=1 em decimal no endereço #8017 corresponde ao valor LSB da variável e, por fim, o valor #00 em 8018 (não aparece na linha acima) corresponde ao valor MSB da variável A%. Como já tínhamos visto, o comando FOR apenas inicializa a variável A% com o primeiro valor, no caso 1, já que não existe um comando NEXT que faça os incrementos sucessivos no valor da variável A%. Para entender um pouco mais sobre este assunto, siga os seguintes passos com o programa DUMP carregado:

1.Digite NEW e pressione a tecla Enter/Return.

2.Digite o seguinte programa em BASIC:

```
10 FOR A%=1 TO 500
20 NEXT A%
```

3.Digite RUN e pressione a tecla Enter/Return.

4.Observe a variação nos endereços #8020 e #8021 (o penúltimo par de valores mostrado pelo DUMP na sua segunda linha)

Que conclusões podemos tirar? Em primeiro lugar, chegamos à conclusão de que o nosso programa DUMP (programa 19) é rápido o suficiente para detectar a variação do conteúdo da variável A%, promovida pelo comando NEXT A%. Em segundo lugar, fica claro que o início da área das variáveis passou para o endereço #801D devido à inclusão da linha 20. Por último, chegamos à conlusão, através dos valores contidos em #8020 e #8021, que a variável A% sai do loop com o valor 5001, e não 5000, como poderíamos pensar. "Mas, como assim?" Ora,

os endereços #8020 e #8021 armazenam o conteúdo da variável A% e o programa DUMP mostra que esses valores são, respectivamente, #89 e #13. Fazendo as contas:

#89=137 em decimal
#13=19 em decimal
Valor final=19*256+137=5001

Você pode conferir tal afirmação entrando com o seguinte comando:

PRINT A%

que o MSX responderá com 5001.

PROCEDIMENTO DE INTERRUPÇÃO DE UM PROGRAMA EM BASIC

Agora que já temos os subsídios necessários, vamos entender como funciona a interpretação de um programa em BASIC. Você já deve saber que um programa em BASIC é interpretado pelo interpretador BASIC do seu MSX, mas, o que vem a ser isso? Um programa **interpretado** tem a sua execução feita linha a linha, ou seja, o interpretador obtém a linha a ser executada, traduz para a linguagem de máquina, executa o programa traduzido e, por fim, retorna com os resultados para o ambiente BASIC. Já um programa **compilado** não apresenta a etapa de tradução; o programa já está todo traduzido para a linguagem de máquina. Agora, fica fácil entender por que os programas compilados são bem mais rápidos do que os programas interpretados. A única vantagem do programa interpretado sobre o compilado é que o primeiro permite uma depuração (etapa de conserto de erros) muito mais fácil, devido à possibilidade de se interromper a execução do programa numa determinada linha. "Mas, Eduardo, como é que se processa a interpretação?" Bem, existe na BIOS do MSX uma rotina chamada **CHRGTR**, que tem como função "varrer" o programa em BASIC na memória. "Varrer? Como assim?" Esta rotina faz um rastreamento da memória, ou seja, quando entramos com um comando RUN ou GOTO, acionamos esta rotina, que é a responsável pela obtenção de cada elemento que constitui uma linha e pelo transporte dele para o tradutor. Como se faz tal rastreamento? O par de registros HL (do Z80) apresenta sempre o endereço de memória do elemento de uma determinada linha que estiver sendo considerado. Diante disto, já devemos desconfiar que, quando entramos com o comando RUN, o par HL é carregado com o endereço #8000, já que este é o endereço inicial de um programa em BASIC. A rotina CHRGTR é muito poderosa, como demonstram as suas características:

Nome da rotina	**: CHRGTR**
Endereço inicial	**: #0010**
Modo de chamada	**: RST #10**
Parâmetros de entrada	**: HL=endereço do caractere atual no programa**
Parâmetros de saída	**: A=próximo caractere no programa**
Registros alterados	**: AF e HL que passa a apontar para o novo caractere**

Tabela 2.1: Características da rotina CHRGTR

Convém mencionar que, no retorno desta rotina, as flags (registro F do Z80) desempenham um papel importante:

Se o próximo caractere for um espaço em branco (#20), tabulação (#09) ou retorno de linha (#0A), será pulado pela rotina CHRGTR.

Se o próximo caractere for um terminador de linha (:) ou fim de linha real (#00), a rotina terminará com as flags Z e NC.

Se o próximo caractere for um dígito entre 0 e 9, ou seja, se for um número, a rotina terminará com as flags NZ e C.

Se o próximo caractere for um caractere puro, ou seja, se não pertencer a nenhum dos tipos anteriores e se não for um token, a rotina terminará com NZ e NC.

Como se vê, além de rastrear o programa em BASIC, esta rotina retorna com o tipo de valor na posição de memória que estiver sendo processada. A união desta rotina com a rotina de interpretação dos erros resultará num modo poderoso e simples de implementação de novos comandos. Para podermos prosseguir na implementação de novos comandos, devemos antes estudar o mecanismo de processamento dos erros.

MECANISMO DE INTERPRETAÇÃO DOS ERROS

À medida que o interpretador BASIC vai "varrendo" o programa em BASIC usando a rotina CHRGTR, coloca, ao mesmo tempo, o código de erro pertinente no registro E do Z80. Se não acontecer nenhum erro, o registro E é carregado com #00; caso contrário, é carregado com o código de erro pertinente. Como já vimos, o erro que nos interessa é o de sintaxe, já que é este o erro que o interpretador BASIC emite quando encontra um comando que desconhece. Consultando o manual que acompanha o nosso MSX, observamos que o código para erro de sintaxe é o #02. O que nos resta fazer? Simples: basta interceptar a rotina de manipulação de erros e verificar se o valor no registro E é igual a #02. "Mas, como vamos interceptar tal rotina?" Usando o hook dessa rotina! Você deve estar lembrado que afirmei que, assim que

uma rotina é ativada, a primeira ação tomada é um desvio para o hook correspondente. Desta forma, o hook é "executado" antes mesmo da própria rotina, abrindo, assim,a chance de termos acesso a todos os parâmetros de entrada dela. No nosso caso, o que nos interessa é ter acesso ao valor contido no registro E, para sabermos se trata-se de um erro de sintaxe ou não. Se for um erro de sintaxe, devemos desviar a rotina de erro para uma rotina nossa que pesquise se o erro foi ou não motivado por um novo comando criado por nós. Se for este o caso, devemos executar as ações pertinentes ao nosso novo comando, devolvendo em seguida o controle ao interpretador BASIC, e não mais à rotina para manipulação de erros. Já se o erro tiver sido motivado por um outro comando que não seja um criado por nós, devemos devolver o controle à rotina para manipulação de erros. Vejamos, então, como se processa todo este mecanismo de desvio, começando por algumas informações básicas e necessárias. A primeira delas se refere ao endereço do hook para desvio da rotina da interpretação de erros. O hook que desejamos chama-se **HERRO** e situa-se no endereço #FFB1. A segunda informação que desejamos diz respeito a como obter o endereço, na memória RAM, do comando que originou o erro. Este endereço é fornecido por uma variável do sistema chamada **SAVTXT**, que se situa no endereço #F6AF. Com estas informações, torna-se possível elaborar uma rotina que crie novos comandos para o BASIC. Mas, antes, uma pergunta: como fazer para que a nossa rotina não gaste um único byte da RAM destinada aos programas em BASIC? Lembre-se que a resposta já foi dada no início deste capítulo. Se você acha que para isso devemos colocar a nossa rotina na página 1 (entre os endereços #4000 e #7FFF), aproveitando a rotina de chamadas interslots, **CALLF**, está absolutamente certo.

O PROGRAMA PARA A CRIAÇÃO DE NOVOS COMANDOS NO BASIC

Com base nas informações expostas nos tópicos anteriores, vamos para a listagem do programa que implementa um novo comando: o comando **REVERSE**. Antes de partir para a listagem, gostaria de fazer um comentário sobre os nomes dos novos comandos. A regra manda que os comandos sejam na mesma língua dos comandos principais, ou seja, em inglês. Eu, particularmente, não dou a mínima para aqueles que, baseados no fato dos comandos estarem em inglês, e não em português, me acusam de ter trocado apenas o nome do verdadeiro autor. Ora, fazer a tradução de um programa é uma tarefa relativamente simples. Então, por que trocar apenas o nome do autor? Se fosse um caso de pirataria, acho que deveria traduzir também os nomes dos comandos, ou não? Aposto que se escrevesse o programa com todos os comandos em português, alguém iria afirmar que fiz o papel de um bom "piratão", traduzindo os nomes dos comandos e trocando o nome do autor verdadeiro! Como se vê, se dermos ouvidos aos argumentos dos incompetentes e invejosos não vamos produzir absolutamente nada. Por que adotei o inglês para dar nome aos novos comandos? A resposta está baseada na coerência, ou seja, se todos os demais comandos são em inglês, por que colocar os novos comandos em português? Outro fato que merece destaque nesta "defesa" é que, sempre que possível, adotarei nomes de comandos já existentes em outras linguagens, como o Turbo Pascal (CLRSCR é um exemplo típico). Agindo dessa forma, estaremos promovendo uma integração maior entre os conceitos adquiridos na experiência com diversas linguagens. Como se

não bastasse, adotando a nomeação em inglês ainda podemos contar com a ajuda dos tokens. Vejamos, por exemplo, o comando **SCRSAVE**. Embora **SCR** não seja um token, **SAVE** o é, de modo que, ao invés do comando acima ocupar 7 bytes, ocupará somente 4. Isto por si só já representa uma boa defesa à nomenclatura em inglês. Vamos então às listagens:

Listagem em assembly Z-80 do código-fonte do programa que implementa o comando REVERSE:

```
;programa para implementação do comando REVERSE
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG22.BIN=PROG22.GEN
;
;onde PROG22.GEN é o nome do arquivo-texto com esta
;listagem

versao      equ     #002d
linlen      equ     #f3b0
txtnam      equ     #f3b3
txtcgp      equ     #f3b7
calslt      equ     #001c
calbas      equ     #0159
chrgtr      equ     #4666
evlexp      equ     #520e
getcoor     equ     #579c
dridef      equ     #f247
crtcnt      equ     #f3b1
errflg      equ     #f414
savtxt      equ     #f6af
scrmod      equ     #fcaf
grpacx      equ     #fcb7
grpacy      equ     #fcb9
hkeyi       equ     #fd9a
herro       equ     #ffb1

            defb    #fe                 ;simula em CP/M
            defw    inicio              ;o cabecalho de
            defw    fim-erro+rotina+#01
                                        ;um arquivo
            defw    inicio              ;.BIN

            org     #d000

inicio
```

```
          di                          ;desabilita as interrupções
          in      a,(#a8)             ;lê a atual configuração dos
          ld      b,a                 ;slots e prepara para ativar
          and     #f0                 ;a página 1 do slot da RAM
          rrca                        ;
          rrca                        ;
          rrca                        ;
          rrca                        ;
          or      b                   ;
          out     (#a8),a             ;ativa as páginas 1, 2 e 3 em
          and     #03                 ;RAM e descobre o slot onde
          ld      (slot),a            ;se encontram os 64Kb de RAM
          ld      a,b                 ;a=config. inicial dos slots
          ld      hl,novohook         ;promove o desvio do hook
          ld      de,herro            ;dos erros
          ld      bc,0005             ;
          ldir                        ;
          ld      hl,rotina           ;transfere a rotina para a
          ld      de,erro             ;página 1 da RAM
          ld      bc,fim-erro+#01     ;
          ldir                        ;
          out     (#a8),a             ;habilita a config. original
          ei                          ;habilita as interrupções
          ret                         ;retorna ao BASIC

novohook
          defb    #f7                 ;RST #30
slot      defb    #00                 ;identificação do slot
          defw    erro                ;endereço de desvio
          defb    #c9                 ;RET

rotina
          org     #4000

erro
          push    af                  ;salva os registros afetados
          push    bc                  ;
          push    de                  ;
          push    hl                  ;
examerro  ld      a,e                 ;a=código do erro
          cp      #02                 ;é erro de sintaxe?
          jp      nz,aborta           ;Não, volta ao BASIC
pegachar  ld      hl,(savtxt)         ;Sim, obtém a posição do
                                      ;ponteiro do BASIC
```

```
            ld      a,(hl)                  ;é final de linha?
            or      a                       ;
            jp      nz,proxchar             ;Não, obtém o próximo carac.
pula        inc     hl                      ;Sim, pula as 4 posições
            inc     hl                      ;referentes ao número da
            inc     hl                      ;linha e ao end. da próxima
            inc     hl                      ;linha
proxchar    ld      ix,chrgtr               ;obtém o primeiro caractere
            call    chamabasic              ;do novo comando
compara     call    compstring              ;compara com a lista de novos
                                            ;comandos
aborta      pop     hl                      ;recupera os registros salvos
            pop     de                      ;
            pop     bc                      ;
            pop     af                      ;
            ret                             ;retorna ao BASIC


;a rotina compstring é a responsável pela localização da
;rotina do novo comando

compstring
            ld      (auxiliar),hl           ;salva o pont. do BASIC
            ld      hl,lista                ;hl aponta p/ lista de
                                            ;comandos
            ld      bc,endlista-lista
                                            ;bc=tamanho da lista
compstr1    ld      de,(auxiliar)           ;de aponta p/ comando em
                                            ;BASIC
            ld      a,(de)                  ;obtém o primeiro byte
            cpir                            ;tenta encontrá-lo na
                                            ;lista
            jp      po,naoachei             ;se bc=0 -> não achou
compstr2    inc     de                      ;achou o primeiro e
                                            ;parte
            ld      a,(de)                  ;para a comparação dos
            cp      (hl)                    ;demais caracteres. Vai
                                            ;para
            jp      nz,compstr1             ;compstr1 se achar um
                                            ;byte diferente.
            inc     hl                      ;Caso contrário,
            ld      a,(hl)                  ;continua até o fim do
                                            ;comando.
            cp      #00                     ;Se chegou ao fim e
            jp      z,achei                 ;então achou o comando
                                            ;na lista.
```

```
               jp      compstr2             ;Caso contrário, compara
naoachei       ret                          ;o próximo byte.
achei          inc     hl                   ;Se achou, obtém o
                                            ;endereço
               ld      c,(hl)               ;de entrada da rotina,
               inc     hl                   ;colocando-o em BC
               ld      b,(hl)               ;
               pop     af                   ;retira da pilha o
                                            ;endereço de
               push    bc                   ;retorno para a rotina
                                            ;erro e
                                            ;coloca na pilha o
                                            ;endereço de
                                            ;entrada da rotina do
                                            ;novo comando
               ld      (auxiliar),de        ;guarda o ponteiro
                                            ;do BASIC
               ex      de,hl                ;transfere o ponteiro
                                            ;para HL
               ret                          ;usa RET para dar um
                                            ;jump
                                            ;para o endereço contido
                                            ;BC que foi salvo na
                                            ;pilha


;a rotina lecomandos é a responsável pela interpretação dos
;valores que se seguem ao nome do novo comando. Os valores
;devem estar separados por vírgulas. Na entrada, o par IX
;aponta para uma tabela com valores default e o registro B
;contém o número de valores a interpretar

lecomandos
               push    ix                   ;salva o par IX
lecoman_1      push    ix                   ;
               ld      ix,chrgtr            ;obtém um valor
               call    chamabasic
               pop     ix                   ;recupera o ponteiro IX
               jr      z,lecoman_4          ;Se não houver valor vai para
                                            ;lecoman_4
               cp      ","                  ;é vírgula?
               jr      z,lecoman_3          ;Sim, assume o default
               dec     hl                   ;Não, obtém o valor
               push    ix                   ;salva o ponteiro
               push    bc                   ;salva o contador
               ld      ix,evlexp            ;obtém o valor
```

```
            call    chamabasic          ;
            pop     bc                  ;recupera o valor
            pop     ix                  ;recupera o ponteiro
            ld      a,e                 ;a=valor interpretado
            ld      (ix+#00),a          ;coloca-o na tabela
lecoman_3   inc     ix                  ;incrementa o ponteiro
            djnz    lecoman_1           ;repete para os demais
                                        ;valores
lecoman_4   pop     ix                  ;recupera o ponteiro salvo
            ret                         ;retorna


;a rotina erro_05 emite a mensagem de chamada ilegal

erro_05
            ld      a,#05               ;a=código do erro
            ld      (errflg),a          ;salva o erro em errflg
            jp      voltaerro           ;vai para voltaerro


;a rotina voltaerro substitui o erro original por um outro

voltaerro
            pop     hl                  ;recupera os pares
            pop     de                  ;salvos no início da
            pop     bc                  ;rotina erro
            pop     af                  ;
            ld      a,(errflg)          ;a=erro a ser emitido
            ld      e,a                 ;e=erro a ser emitido
            ret                         ;volta para a rotina
                                        ;de manipulação de erros
                                        ;do interpretador BASIC


;a rotina volta promove a volta ao BASIC sem que haja
;interrupção no processamento, ou seja, sem que o sistema
;detecte o erro ocorrido

volta
            dec     hl                  ;hl aponta para o final
                                        ;do último comando
            xor     a                   ;modifica para nenhum erro
            ld      (errflg),a          ;ocorrido
            pop     de                  ;recupera os registros
```

```
            pop     de                  ;salvos no início da rotina
            pop     bc                  ;erro. Note que hl não é
            pop     af                  ;recuperado.
            pop     de                  ;obtém o end. de retorno da
                                        ;nossa rotina para a rotina
                                        ;de chaveamento de slots
            pop     bc                  ;obtém o end. de retorno da
                                        ;rotina de chaveamento de
                                        ;slots para RST #30
            pop     af                  ;obtém o end. de retorno de
                                        ;RST #30 para o hook
            pop     ix                  ;obtém/destrói o end. de
                                        ;retorno do hook para a
                                        ;rotina
                                        ;de erro do interpretador
                                        ;BASIC
            pop     ix                  ;obtém/destrói o end. de
                                        ;retorno
                                        ;da rotina de erro para o
                                        ;interpretador BASIC
            push    af                  ;repõe na pilha os end. de
            push    bc                  ;retorno salvos nestes
            push    de                  ;registros
            ld      ix,chrgtr           ;faz com que hl aponte
            jp      chamabasic          ;para o próximo comando
            ret                         ;e retorna ao BASIC

chamabasic
            call    calbas              ;chama a rotina calbas
            ret                         ;retorna

inverte
            ld      b,#04               ;b=núm. de valores
            ld      ix,tabinver         ;ix=tabela default
            call    lecomandos          ;lê os 4 valores
            ld      a,(ix+#00)          ;obtém coord. x
            ld      (grpacx),a          ;salva em grpacx
            ld      a,(ix+#01)          ;obtém a coord. y
            ld      (grpacy),a          ;salva em grpacy
            ld      a,(ix+#02)          ;obtém o comp. da string
            ld      (compstr),a         ;salva em compstr
            cp      33                  ;é >= 33?
            jp      nc,erro_05          ;Sim, chamada ilegal
            ld      (auxiliar),hl       ;salva ptr. do BASIC
            ld      a,(ix+#03)          ;a=flag de recuperação
            or      a                   ;está setada?
```

```
          jr      z,inverte0           ;Não, vai para inverte0
          ld      a,(inversao)         ;Sim, já foi feita alguma
          or      a                    ;inversão?
          jr      z,inverte0           ;Não, vai para inverte0
          ld      hl,(bufnor+#01)      ;Sim, obtém a pos. da string
          call    setvdpwt             ;antiga e prepara o VDP
          ld      hl,bufnor+#03        ;hl->início da string antiga
          ld      a,(bufnor)           ;a=comp. da string antiga
          ld      b,a                  ;a=comp. da string antiga
          call    esclinha             ;escreve a string antiga
                                       ;no modo normal
inverte0  ld      a,#ff                ;
          ld      (inversao),a         ;sinaliza uma inversão
          ld      hl,(auxiliar)        ;hl=ponteiro do BASIC
          ld      a,(compstr)          ;a=comp. da string
          or      a                    ;comprimento=0?
          jp      z,volta              ;Sim, volta com erro
          ld      ix,bufnor            ;Não, ajusta os buffers
          ld      iy,bufinv            ;para a string normal e
          ld      (ix+#00),a           ;para a string invertida
          ld      (iy+#00),a           ;com o comprimento da
          call    calpadvid            ;calcula os padrões do vídeo
          ld      a,(grpacx)           ;obtém a coord. x
          ld      e,a                  ;coloca em e
          ld      a,(colunas)          ;obtém o número de colunas
          cp      e                    ;compara com o valor lido
          jp      c,erro_05            ;Se valor lido > colunas
                                       ;volta com erro
          ld      a,(grpacy)           ;obtém a coord. y
          cp      24                   ;é maior do que 23?
          jp      nc,erro_05           ;Sim, volta com erro
          call    lefrase              ;lê a string a inverter
inverte3  di                           ;desabilita as interrupções
          ld      hl,(tabpad)          ;hl=end. da tab. de padrões
          push    hl                   ;salva o end. da tab. de
                                       ;padrões
          ld      de,#e0*8             ;calcula o endereço do
          add     hl,de                ;desenho do caractere #e0
          ex      de,hl                ;de=end. do des. do carac. #e0
          ld      c,#e0                ;c=#e0
          ld      b,(ix+#00)           ;b=número de carac. da string
          ld      ix,bufnor+#03        ;ix=endereço da string normal
          ld      iy,bufinv+#03        ;iy=end. da string invertida
          pop     hl                   ;recupera hl
loopinv1  push    bc                   ;salva bc
          push    hl                   ;salva hl
          push    de                   ;salva de
```

```
                ld      de,#0008            ;bytes para o desenho de cada
                                            ;caractere
                ld      b,(ix+#00)          ;b=caractere a inverter
loopinv2        add     hl,de               ;calcula o endereço desse
                djnz    loopinv2            ;caractere
                pop     de                  ;recupera de
                ld      b,#08               ;prepara a modificação
loopinv3        call    rdvram              ;lê o byte original
                cpl                         ;inverte
                ex      de,hl               ;transfere para o novo
                call    wtvram              ;destino
                ex      de,hl               ;
                inc     hl                  ;hl=hl+1
                inc     de                  ;de=de+1
                djnz    loopinv3            ;b=b-1
                pop     hl                  ;recupera hl
                pop     bc                  ;recupera bc
                ld      (iy+#00),c          ;modifica a string
                inc     c                   ;c=c+1
                inc     ix                  ;aponta para o próximo carac.
                inc     iy
                djnz    loopinv1            ;repete até o fim da string
envinvert
                ld      iy,bufinv           ;recupera o pont. do buffer
                ld      l,(iy+#01)          ;hl=end. de escrita da string
                ld      h,(iy+#02)          ;
                call    setvdpwt            ;ajusta o VDP para escrita
                ld      b,(iy+#00)          ;b=núm. de carac. da string
                ld      hl,bufinv+#03       ;end. da string
                call    esclinha            ;escreve a frase invertida
                ei                          ;habilita as interrupções
                ld      hl,(auxiliar)       ;recupera ptr. do BASIC
                jp      volta               ;volta para o BASIC


tabinver
                defb    #00                 ;posição x
                defb    #00                 ;posição y
                defb    #00                 ;comprimento
                defb    #00                 ;flag de restauração



;a rotina calpadvid calcula os padrões do vídeo: a tabela de
;padrões e o número de colunas. Os valores resultantes são
;colocados em tabpad e colunas, respectivamente
```

```
calpadvid
            ex      af,af'            ;salva o registro af
            exx                       ;salva os demais registros
            ld      hl,(txtcgp)       ;obtém a tab. de padrões
            ld      a,(versao)        ;obtém a versão
            or      a                 ;do MSX. É MSX1?
            jr      z,calpad_1        ;Sim, vai para calpad_1
            ld      a,(linlen)        ;obtém o tamanho da tela
            cp      41                ;O MSX 2 está em 80 colunas?
            jr      c,calpad_1        ;Não, vai para calpad_1
            ld      a,80              ;Sim, a=80 colunas
            add     hl,hl             ;calcula novo end. da
                                      ;tabela de padrões
            jr      calpad_2          ;vai para calpad_2
calpad_1    ld      a,40              ;a=40 colunas
calpad_2    ld      (colunas),a       ;salva as colunas
            ld      (tabpad),hl       ;salva o end. da tab. de
                                      ;padrões
            exx                       ;recupera os registros salvos
            ex      af,af'            ;
            ret                       ;retorna


;a rotina lefrase transporta da tela para o buffer bufnor
;a sentença a inverter.
lefrase
            push    af                ;salva os registros
            push    bc                ;modificados por esta
            push    de                ;rotina
            push    hl                ;
            call    calendfra         ;calcula o end. da string
                                      ;na VRAM
            call    setvdprd          ;prepara o VDP para leitura
            ld      b,(ix+#00)        ;obtém o comp. da string
            ld      hl,bufnor+#03     ;hl->início da string
            call    lelinha           ;lê a string
            pop     hl                ;recupera os registros
            pop     de                ;salvos
            pop     bc                ;
            pop     af                ;
            ret                       ;retorna


;a rotina calendfra,baseada nas coordenadas fornecidas, calcula o endereço da sentença a
;inverter na memória VRAM
```

```
calendfra
              ld      h,#00              ;calcula o end. da string
              ld      d,h                ;na memória VRAM, baseada
              ld      a,(grpacy)         ;nas coordenadas passadas
              ld      l,a                ;
              or      a                  ;a string está na linha 0?
              jr      z,calend1          ;Sim, vai para calend1
              ld      l,h                ;Não, calcula o end. do
              ld      b,a                ;início da linha
              ld      a,(colunas)        ;
              ld      e,a                ;
loopcal_1     add     hl,de              ;
              djnz    loopcal_1          ;
calend1       ld      a,(grpacx)         ;obtém a coord. x
              ld      e,a                ;e=coord. x
              add     hl,de              ;soma ao end. já encontrado
              ld      (bufnor+#01)       ,hl;salva no buffer das strings
              ld      (bufinv+#01)       ,hl;normal e invertida
              ret                        ;retorna


;início da lista com os nomes dos novos comandos e
;endereços de chamada

lista
              defm    "REVERSE"          ;nome do comando
              defb    #00                ;fim do nome
              defw    inverte            ;end. da rotina
endlista      defb    #00                ;fim da lista


;rotinas para o acesso direto à RAM de vídeo

rdvram
              call    setvdprd           ;ajusta o VDP para leitura
              in      a,(#98)            ;lê um byte
              ret                        ;retorna


wtvram
              push    af                 ;salva o dado a enviar
              call    setvdpwt           ;ajusta o VDP para escrita
              pop     af                 ;recupera o dado a enviar
              out     (#98),a            ;envia
              ret                        ;retorna
```

```
setvdprd
            ld      a,(versao)          ;obtém a versão do MSX
            or      a                   ;é MSX1?
            jr      z,rdvram1           ;Sim, vai para rdvram1
            xor     a                   ;Não, inicializa o VDP
            out     (#99),a             ;do MSX2
            ld      a,#8e
            out     (#99),a             ;
rdvram1     ld      a,l                 ;informa ao
            out     (#99),a             ;VDP o endereço na
            ld      a,h                 ;VRAM onde será
            and     #3f                 ;lido o dado
            out     (#99),a             ;
            ex      (sp),hl             ;demora para
            ex      (sp),hl             ;sincronização
            ret                         ;retorna


setvdpwt
            ld      a,(versao)          ;obtém a versão do MSX
            or      a                   ;é MSX1?
            jr      z,wtvram1           ;Sim, vai para wtvram1
            xor     a                   ;Não, inicializa o VDP
            out     (#99),a             ;do MSX2
            ld      a,#8e
            out     (#99),a
wtvram1     ld      a,l                 ;informa ao
            out     (#99),a             ;VDP o endereço na
            ld      a,h                 ;VRAM onde o
            and     #3f                 ;dado será
            or      #40                 ;gravado
            out     (#99),a             ;
            ex      (sp),hl             ;demora para
            ex      (sp),hl             ;sincronização
            ret                         ;retorna


;a rotina lelinha transporta uma linha da VRAM para a RAM

lelinha
            in      a,(#98)             ;lê o carac. da VRAM
            ld      (hl),a              ;salva-o no buffer
            inc     hl                  ;incrementa o ponteiro
            djnz    lelinha             ;prepara a próxima leitura
            ret                         ;retorna
```

```
;a rotina esclinha transporta uma linha da RAM para a VRAM

esclinha
            ld      a,(hl)          ;lê o carac. do buffer
            out     (#98),a         ;escreve-o na VRAM
            inc     hl              ;incrementa o ponteiro
            djnz    esclinha        ;prepara a próxima escrita
            ret                     ;retorna

;área das variáveis usadas no programa

bufnor
            defb    #00             ;tamanho da string
            defw    #0000           ;posição de escrita
            defs    33              ;string

bufinv
            defb    #00             ;tamanho da string
            defw    #0000           ;posição de escrita
            defs    33              ;string

inversao    defb    #00             ;flag de campo já invertido

compstr     defb    #00             ;comprimento da string

colunas     defb    #00             ;número de colunas na tela

tabpad      defw    #0000           ;endereço da tab. de padrões

auxiliar    defw    #0000           ;ponteiro do BASIC

fim         equ     $
```

Listagem em linhas DATA do código-objeto do programa que implementa o comando REVERSE:

```
10 FOR A%=&HD000 TO &HD2BB
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "prog22.BIN",&HD000,&HD2BB
100 DATA F3,DB,A8,47,E6,F0,0F,0F,0F,0F,B0,D3,A8,E6,03,32
110 DATA 2E,D0,78,21,2D,D0,11,B1,FF,01,05,00,ED,B0,21,32
120 DATA D0,11,00,40,01,89,02,ED,B0,D3,A8,FB,C9,F7,00,00
130 DATA 40,C9,F5,C5,D5,E5,7B,FE,02,C2,20,40,2A,AF,F6,7E
```

```
140 DATA B7,C2,16,40,23,23,23,23,DD,21,66,46,CD,AD,40,CD
150 DATA 25,40,E1,D1,C1,F1,C9,22,86,42,21,E0,41,01,0A,00
160 DATA ED,5B,86,42,1A,ED,B1,E2,48,40,13,1A,BE,C2,2E,40
170 DATA 23,7E,FE,00,CA,49,40,C3,38,40,C9,23,4E,23,46,F1
180 DATA C5,ED,53,86,42,EB,C9,DD,E5,DD,E5,DD,21,66,46,CD
190 DATA AD,40,DD,E1,28,1A,FE,2C,28,12,2B,DD,E5,C5,DD,21
200 DATA 0E,52,CD,AD,40,C1,DD,E1,7B,DD,77,00,DD,23,10,D9
210 DATA DD,E1,C9,3E,05,32,14,F4,C3,89,40,E1,D1,C1,F1,3A
220 DATA 14,F4,5F,C9,2B,AF,32,14,F4,D1,D1,C1,F1,D1,C1,F1
230 DATA DD,E1,DD,E1,F5,C5,D5,DD,21,66,46,C3,AD,40,C9,CD
240 DATA 59,01,C9,06,04,DD,21,83,41,CD,55,40,DD,7E,00,32
250 DATA B7,FC,DD,7E,01,32,B9,FC,DD,7E,02,32,82,42,FE,21
260 DATA D2,81,40,22,86,42,DD,7E,03,B7,28,16,3A,81,42,B7
270 DATA 28,10,2A,3A,42,CD,11,42,21,3C,42,3A,39,42,47,CD
280 DATA 32,42,3E,FF,32,81,42,2A,86,42,3A,82,42,B7,CA,92
290 DATA 40,DD,21,39,42,FD,21,5D,42,DD,77,00,FD,77,00,CD
300 DATA 87,41,3A,B7,FC,5F,3A,83,42,BB,DA,81,40,3A,B9,FC
310 DATA FE,18,D2,81,40,CD,A9,41,F3,2A,84,42,E5,11,00,07
320 DATA 19,EB,0E,E0,DD,46,00,DD,21,3C,42,FD,21,60,42,E1
330 DATA C5,E5,D5,11,08,00,DD,46,00,19,10,FD,D1,06,08,CD
340 DATA EB,41,2F,EB,CD,F1,41,EB,23,13,10,F3,E1,C1,FD,71
350 DATA 00,0C,DD,23,FD,23,10,D8,FD,21,5D,42,FD,6E,01,FD
360 DATA 66,02,CD,11,42,FD,46,00,21,60,42,CD,32,42,FB,2A
370 DATA 86,42,C3,92,40,00,00,00,00,08,D9,2A,B7,F3,3A,2D
380 DATA 00,B7,28,0C,3A,B0,F3,FE,29,38,05,3E,50,29,18,02
390 DATA 3E,28,32,83,42,22,84,42,D9,08,C9,F5,C5,D5,E5,CD
400 DATA C1,41,CD,F9,41,DD,46,00,21,3C,42,CD,2B,42,E1,D1
410 DATA C1,F1,C9,26,00,54,3A,B9,FC,6F,B7,28,09,6C,47,3A
420 DATA 83,42,5F,19,10,FD,3A,B7,FC,5F,19,22,3A,42,22,5E
430 DATA 42,C9,52,45,56,45,52,53,45,00,B1,40,00,CD,F9,41
440 DATA DB,98,C9,F5,CD,11,42,F1,D3,98,C9,3A,2D,00,B7,28
450 DATA 07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,D3,99
460 DATA E3,E3,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99
470 DATA 7D,D3,99,7C,E6,3F,F6,40,D3,99,E3,E3,C9,DB,98,77
480 DATA 23,10,FA,C9,7E,D3,98,23,10,FA,C9,00,00,00,00,00
490 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
500 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
510 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
520 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
530 DATA 00,00,00,00,00,00,00,00,00,00,00,09
```

Listagem do programa de teste:

```
100 CLS:DIM B$(5):BLOAD"PROG22.BIN",R
110 REM *** Verifica o tamanho da tela ***
120 IF PEEK(&HF3B0)>40 THEN CP%=80 ELSE CP%=40
```

```
130 REM *** Loop de leitura dos comandos ***
140 FOR A%=1 TO 5
150 READ B$(A%)
160 NEXT A%
170 REM *** Loop para imprimir as opções ***
180 FOR A%=1 TO 5
190 LOCATE (CP%-LEN(B$(A%)))/2,A%+5:PRINTB$(A%)
200 NEXT A%
210 REM *** Loop de demonstração do comando REVERSE ***
220 SL%=1:PS%=6
230 X%=(CP%-LEN(B$(SL%)))/2:CM%=LEN(B$(SL%)):REVERSE X%,PS%,CM%,1
240 A$=INKEY$:IF A$="" THEN 240
250 AT%=ASC(A$)
260 IF (AT%<>&H1E AND AT%<>&H1F) THEN 240
270 IF (AT%=&H1E AND PS%>6) THEN PS%=PS%-1:SL%=SL%-1:GOTO 230
280 IF (AT%=&H1E AND PS%=6) THEN PS%=10:SL%=5:GOTO 230
290 IF (AT%=&H1F AND PS%<10) THEN PS%=PS%+1:SL%=SL%+1:GOTO 230
300 GOTO 220
310 REM *** Comandos a serem exibidos ***
320 DATA Banco de dados,Planilha,Editor de textos,Editor gráfico,Sair
```

COMENTÁRIOS SOBRE O PROGRAMA QUE IMPLEMENTA O COMANDO REVERSE

Antes de executar o programa de teste, vamos rever alguns tópicos. Antes de mais nada, vamos começar pelo estudo da sintaxe do novo comando. A sintaxe é:

REVERSE ***X,Y,Comp,Flag***

onde

X é a coordenada horizontal do primeiro caractere da sentença a ser invertida

Y é a coordenada vertical do primeiro caractere da sentença a ser invertida

Comp é o comprimento, em caracteres, da sentença a ser invertida

Flag é um indicador que ativa (1) ou desativa (0) a reinversão da última sentença

Uma vez vista a sintaxe, vamos para a análise do programa-fonte.

Antes de analisarmos as rotinas criadas por nós, vamos analizar duas rotinas presentes na ROM do MSX e por nós usadas: **CALBAS** e **EVLEXP**. A rotina **CALBAS** tem como função chamar uma rotina do interpretador BASIC a partir de qualquer configuração de slots. Esta rotina se torna particularmente útil porque o interpretador BASIC ocupa as páginas 0 e 1 do slot

0, e a nossa rotina ocupa a página 1 do slot da RAM. Como duas páginas iguais (no caso a 1, que é usada tanto pelo interpretador BASIC como pelo nosso programa, ainda que em slots diferentes) não podem ser ativadas ao mesmo tempo, só nos resta o recurso de utilizar a rotina **CALBAS** que, em síntese, desativa a página 1 do slot atual, ativa a página 1 do slot 0 (ROM), executa a rotina desejada, desativa a página 1 do slot 0 e, por fim, ativa a página 1 do slot atual. Como você pode comprovar, é uma rotina muito versátil e útil. Já a rotina **EVLEXP** (do inglês evaluate expression - calcular a expressão) tem como função calcular o resultado final de uma expressão em BASIC, tal como 2*2, X+Y, X+LEN(B$), etc. Esta rotina retorna com o resultado calculado como um inteiro no par DE. A Tabela 2.2 apresenta as características destas duas rotinas.

Nome da rotina	**: CALBAS**
Endereço inicial	**: #0159**
Modo de chamada	**: CALL**
Parâmetros de entrada	**: IX=endereço da rotina BASIC a ser executada**
Parâmetros de saída	**: Nenhum**
Registros alterados	**: AF',BC',DE',HL',IY**
Nome da rotina	**: EVLEXP**
Endereço inicial	**: #520E**
Modo de chamada	**: Por intermédio de CALBAS**
Parâmetros de entrada	**: HL=início da expressão a avaliar**
Parâmetros de saída	**: HL=fim da expressão avaliada DE=valor inteiro do resultado**
Registros alterados	**: AF, BC, DE, HL**

Tabela 2.2: Características das rotinas CALBAS e EVLEXP

A rotina ***lecomandos*** foi projetada para interpretar comandos separados por vírgula, permitindo inclusive que o usuário não entre com determinados valores. Por exemplo, se você desejar reinverter (colocar no modo normal) a última sentença invertida, bastará entrar com o comando

REVERSE ,,0,1

onde as duas primeiras vírgulas provocam a adoção dos valores default (no caso, os últimos usados) para x e y. Como o terceiro parâmetro é o comprimento da string a inverter, basta colocá-lo em zero para que não haja inversão alguma. Como você já sabe, o quarto e último parâmetro define a reinversão da última sentença invertida. Como afirmam os comentários no cabeçalho desta rotina, o par IX na entrada deve apontar para uma tabela de valores default que será preenchida ou não, de acordo com os valores fornecidos pelo usuário. Você pode não estar vendo muito sentido nesta rotina, já que só existe um comando implementado. Ocorre que a versão final deste programa apresenta mais de vinte comandos, logo, seria ilógico possuir uma rotina de interpretação para cada um deles. Como quase todos os

comandos utilizam a entrada de valores separados por vírgulas, senti-me na obrigação de fazer uma rotina geral para interpretação de valores. Imagine a economia de bytes que isto gerou! Além da rotina ***lecomandos***, temos uma outra rotina que, devido à técnica empregada, pode não ter o seu funcionamento perfeitamente entendido. Vamos, então, ao estudo da rotina ***volta***. Essa rotina é a responsável por enganar o interpretador BASIC quanto ao erro de sintaxe. Vamos ver como se encontra a pilha após a execução da rotina que implementa o comando REVERSE:

Bytes	**Motivo**
2	End. de retorno da rotina de erros para o interpretador BASIC
2	End. de retorno do hook para a rotina de erros
2	Par AF salvo pela rotina ***erro***
2	Par BC salvo pela rotina ***erro***
2	Par DE salvo pela rotina ***erro***
2	Par HL salvo pela rotina ***erro***

Como não desejamos que o sistema recorde que ocorreu um erro de sintaxe, temos de destruir os quatro bytes do topo da pilha referentes aos endereços de retorno usados pela rotina de erro. Por outro lado, temos de recuperar os registros salvos na pilha (AF, BC, DE e HL). "Como fazer, então?" A resposta mais simples é fazer seis pops seguidos de quatro pushs, como mostra a listagem do código-fonte. Cabe aqui um lembrete: nunca utilize um comando que use o desvio do vetor de erro (como o comando REVERSE) em conjunto com o comando **IF...THEN**. Por exemplo, o comando

IF A%<9 THEN REVERSE 0,0,10,1

resultaria numa mensagem de erro de sintaxe. O motivo é simples: o comando **THEN** faz uma avaliação do próximo comando antes de executá-lo, logo, temos neste caso dois indicadores de erro, o do comando **THEN** e o do interpretador BASIC ao executar o comando **REVERSE**. Como o hook que usamos só atua sobre os erros de sintaxe vistos pelo interpretador BASIC, não temos como destruir a indicação de erro de sintaxe colocada na pilha pelo comando THEN. "Como posso evitar essa limitação?" Onde for possível, basta que você inverta as situações. Veja, por exemplo, a linha a seguir:

110 IF A%<9 THEN REVERSE 0,A%,10,1:ELSE A%=A%+1

Ela poderá ser transformada em

110 IF A%>=9 THEN A%=A%+1 ELSE REVERSE 0,A%,10,1

sem qualquer prejuízo ao processamento. Além desta simples inversão na ordem dos comandos, você sempre poderá lançar mão dos comandos **GOSUB** (preferível) e **GOTO**. Vamos, então, ver algumas das características deste tipo de implementação.

Pontos positivos

1. Não faz uso de um byte sequer da RAM destinada ao BASIC
2. Não faz uso do comando USR, que, por não ser descritivo, é confuso
3. É totalmente transparente, dando a sensação de que o comando está disponível na ROM do microcomputador
4. Permite a emissão de novos erros, como os erros de tipos não compatíveis (mismatch error), de chamada ilegal (illegal function call), etc.
5. Permite o uso de funções do próprio BASIC, como, por exemplo, em:

   REVERSE X, Y, LEN(B$(A%)),0

Ponto negativo

1. Não pode ser usado após um comando **THEN**

Tabela 2.3: Características dos comandos implementados por desvio do vetor de erro

O PROGRAMA QUE IMPLEMENTA O COMANDO CLRSCR

Como afirmei no início deste capítulo, terminaríamos este tópico com a implementação do comando **CLRSCR**. Você vai perceber que este novo comando nada mais é do que uma união de 5 programas do Capítulo 1 com o programa que implementa o comando **REVERSE**. Tendo em vista que as listagens permanecem praticamente inalteradas, creio que você não terá qualquer dificuldade em acompanhar a lógica envolvida. Vamos então às listagens.

Listagem em assembly Z-80 do código-fonte do programa que implementa o comando CLRSCR:

```
;programa para implementar o comando CLRSCR
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
```

```
;
;GEN80 PROG23.BIN=PROG23.GEN
;
;onde PROG23.GEN é o nome do arquivo-texto com esta
;listagem

versao      equ     #002d
linlen      equ     #f3b0
txtnam      equ     #f3b3
txtcgp      equ     #f3b7
calslt      equ     #001c
calbas      equ     #0159
chrgtr      equ     #4666
evlexp      equ     #520e
getcoor     equ     #579c
dridef      equ     #f247
crtcnt      equ     #f3b1
errflg      equ     #f414
savtxt      equ     #f6af
scrmod      equ     #fcaf
grpacx      equ     #fcb7
grpacy      equ     #fcb9
hkeyi       equ     #fd9a
herro       equ     #ffb1

            defb    #fe                 ;simula em CP/M
            defw    inicio              ;o cabecalho de
            defw    fim-erro+rotina+#01
                                        ;um arquivo
            defw    inicio              ;.BIN


            org     #d000

inicio
            di                          ;desabilita as interrupções
            in      a,(#a8)             ;lê a atual configuração dos
            ld      b,a                 ;slots e prepara para ativar
            and     #f0                 ;a página 1 do slot da RAM
            rrca                        ;
            rrca
            rrca                        ;
            rrca                        ;
            or      b                   ;
            out     (#a8),a             ;ativa as páginas 1, 2 e 3 em
            and     #03                 ;RAM e descobre o slot onde
            ld      (slot),a            ;se encontram os 64Kb de RAM
```

```
            ld      a,b              ;a=config. inicial dos slots
            ld      hl,novohook      ;promove o desvio do hook
            ld      de,herro         ;dos erros
            ld      bc,0005          ;
            ldir                     ;
            ld      hl,rotina        ;transfere a rotina para a
            ld      de,erro          ;página 1 da RAM
            ld      bc,fim-erro+#01  ;
            ldir                     ;
            out     (#a8),a          ;habilita a config. original
            ei                       ;habilita as interrupções
            ret                      ;retorna ao BASIC

novohook
            defb    #f7              ;RST #30
slot        defb    #00              ;identificação do slot
            defw    erro             ;endereço de desvio
            defb    #c9              ;RET

rotina
            org     #4000

erro
            push    af               ;salva os registros afetados
            push    bc               ;
            push    de               ;
            push    hl               ;
examerro    ld      a,e              ;a=código do erro
            cp      #02              ;é erro de sintaxe?
            jp      nz,aborta        ;Não, volta ao BASIC
pegachar    ld      hl,(savtxt)      ;Sim, obtém a posição do
                                     ;ponteiro do BASIC
            ld      a,(hl)           ;é final de linha?
            or      a                ;
            jp      nz,proxchar      ;Não, obtém o próximo carac.
pula        inc     hl               ;Sim, pula as 4 posições
            inc     hl               ;referentes ao número da
            inc     hl               ;linha e ao end. da próxima
            inc     hl               ;linha
proxchar    ld      ix,chrgtr        ;obtém o primeiro caractere
            call    chamabasic       ;do novo comando
compara     call    compstring       ;compara com a lista de novos
                                     ;comandos
aborta      pop     hl               ;recupera os registros salvos
```

```
                pop     de                ;
                pop     bc                ;
                pop     af                ;
                ret                       ;retorna ao BASIC
```

;a rotina compstring é a responsável pela localização da
;rotina do novo comando

```
compstring
                ld      (auxiliar),hl     ;salva o pont. do BASIC
                ld      hl,lista          ;hl aponta p/ lista de
                                          ;comandos
                ld      bc,endlista-lista
                                          ;bc=tamanho da lista
compstr1        ld      de,(auxiliar)     ;de aponta p/ comando em
                                          ;BASIC
                ld      a,(de)            ;obtém o primeiro byte
                cpir                      ;tenta encontrá-lo na
                                          ;lista
                jp      po,naoachei       ;se bc=0 -> não achou
compstr2        inc     de                ;achou o primeiro e
                                          ;parte
                ld      a,(de)            ;para a comparação dos
                cp      (hl)              ;demais caracteres. Vai
                                          ;para
                jp      nz,compstr1       ;compstr1 se achar um
                                          ;byte diferente.
                inc     hl                ;Caso contrário,
                ld      a,(hl)            ;continua até o fim do
                                          ;comando.
                cp      #00               ;Se chegou ao fim e,
                jp      z,achei           ;então, achou o comando
                                          ;na lista.
                jp      compstr2          ;Caso contrário, compara
naoachei        ret                       ;o próximo byte.
achei           inc     hl                ;Se achou, obtém o
                                          ;endereço
                ld      c,(hl)            ;de entrada da rotina
                inc     hl                ;colocando-o em BC
                ld      b,(hl)            ;
                pop     af                ;retira da pilha o
                                          ;endereço de
                push    bc                ;retorno para a rotina
                                          ;erro e
                                          ;coloca na pilha o
```

```
                                  ;endereço de
                                  ;entrada da rotina do
                                  ;novo comando
          ld      (auxiliar),de   ;guarda o ponteiro
                                  ;do BASIC
          ex      de,hl           ;transfere o ponteiro
                                  ;para HL
          ret                     ;usa RET para dar um
                                  ;jump
                                  ;para o endereço contido
                                  ;BC que foi salvo na
                                  ;pilha


;a rotina lecomandos é a responsável pela interpretação dos
;valores que se seguem ao nome do novo comando. Os valores
;devem estar separados por vírgulas. Na entrada, o par IX
;aponta para uma tabela com valores default e o registro B
;contém o número de valores a interpretar

lecomandos
          push    ix              ;salva o par IX
lecoman_1 push    ix              ;
          ld      ix,chrgtr       ;obtém um valor
          call    chamabasic
          pop     ix              ;recupera o ponteiro IX
          jr      z,lecoman_4     ;Se não houver valor, vai para
                                  ;lecoman_4
          cp      ","             ;é vírgula?
          jr      z,lecoman_3     ;Sim, assume o default
          dec     hl              ;Não, obtém o valor
          push    ix              ;salva o ponteiro
          push    bc              ;salva o contador
          ld      ix,evlexp       ;obtém o valor
          call    chamabasic      ;
          pop     bc              ;recupera o valor
          pop     ix              ;recupera o ponteiro
          ld      a,e             ;a=valor interpretado
          ld      (ix+#00),a      ;coloca-o na tabela
lecoman_3 inc     ix              ;incrementa o ponteiro
          djnz    lecoman_1       ;repete para os demais
                                  ;valores
lecoman_4 pop     ix              ;recupera o ponteiro salvo
          ret                     ;retorna
```

```
;a rotina erro_05 emite a mensagem de chamada ilegal

erro_05
            ld      a,#05           ;a=código do erro
            ld      (errflg),a      ;salva o erro em errflg
            jp      voltaerro       ;vai para voltaerro


;a rotina voltaerro substitui o erro original por um outro

voltaerro
            pop     hl              ;recupera os pares
            pop     de              ;salvos no início da
            pop     bc              ;rotina erro
            pop     af              ;
            ld      a,(errflg)      ;a=erro a ser emitido
            ld      e,a             ;e=erro a ser emitido
            ret                     ;volta para a rotina
                                    ;de manipulação de erros
                                    ;do interpretador BASIC


;a rotina volta promove a volta ao BASIC sem que haja
;interrupção no processamento, ou seja, sem que o sistema
;detecte o erro ocorrido

volta
            dec     hl              ;hl aponta para o final
                                    ;do último comando
            xor     a               ;modifica para nenhum erro
            ld      (errflg),a      ;ocorrido
            pop     de              ;recupera os registros
            pop     de              ;salvos no início da rotina
            pop     bc              ;erro. Note que hl não é
            pop     af              ;recuperado.
            pop     de              ;obtém o end. de retorno da
                                    ;nossa rotina para a rotina
                                    ;de chaveamento de slots
            pop     bc              ;obtém o end. de retorno da
                                    ;rotina de chaveamento de
                                    ;slots para RST #30
            pop     af              ;obtém o end. de retorno de
                                    ;RST #30 para o hook
            pop     ix              ;obtém/destrói o end. de
                                    ;retorno do hook para a
```

```
                                ;rotina
                                ;de erro do interpretador
                                ;BASIC
           pop     ix           ;obtém/destrói o end. de
                                ;retorno
                                ;da rotina de erro para o
                                ;interpretador BASIC
           push    af           ;repõe na pilha os end. de
           push    bc           ;retorno salvos nestes
           push    de           ;registros
           ld      ix,chrgtr    ;faz com que hl aponte
           jp      chamabasic   ;para o próximo comando
           ret                  ;e retorna ao BASIC


chamabasic
           call    calbas       ;chama a rotina calbas
           ret                  ;retorna

inverte
           ld      b,#04        ;b=núm. de valores
           ld      ix,tabinver  ;ix=tabela default
           call    lecomandos   ;lê os 4 valores
           ld      a,(ix+#00)   ;obtém a coord. x
           ld      (grpacx),a   ;salva em grpacx
           ld      a,(ix+#01)   ;obtém a coord. y
           ld      (grpacy),a   ;salva em grpacy
           ld      a,(ix+#02)   ;obtém o comp. da string
           ld      (compstr),a  ;salva em compstr
           cp      33           ;é >= 33?
           jp      nc,erro_05   ;Sim, chamada ilegal
           ld      (auxiliar),hl ;salva ptr. do BASIC
           ld      a,(ix+#03)   ;a=flag de recuperação
           or      a            ;está setada?
           jr      z,inverte0   ;Não, vai para inverte0
           ld      a,(inversao) ;Sim, já foi feita alguma
           or      a            ;inversão?
           jr      z,inverte0   ;Não, vai para inverte0
           ld      hl,(bufnor+#01) ;Sim, obtém a pos. da string
           call    setvdpwt     ;antiga e prepara o VDP
           ld      hl,bufnor+#03 ;hl->início da string antiga
           ld      a,(bufnor)   ;a=comp. da string antiga
           ld      b,a          ;a=comp. da string antiga
           call    esclinha     ;escreve a string antiga
                                ;no modo normal
inverte0   ld      a,#ff        ;
           ld      (inversao),a ;sinaliza uma inversão
```

```
          ld      hl,(auxiliar)     ;hl=ponteiro do BASIC
          ld      a,(compstr)       ;a=comp. da string
          or      a                 ;comprimento=0?
          jp      z,volta           ;Sim, volta com erro
          ld      ix,bufnor         ;Não, ajusta os buffers
          ld      iy,bufinv         ;para a string normal e
          ld      (ix+#00),a        ;para a string invertida
          ld      (iy+#00),a        ;com o comprimento da
          call    calpadvid         ;calcula os padrões do vídeo
          ld      a,(grpacx)        ;obtém a coord. x
          ld      e,a               ;coloca em e
          ld      a,(colunas)       ;obtém o número de colunas
          cp      e                 ;compara com o valor lido
          jp      c,erro_05         ;Se valor lido > colunas
                                    ;volta com erro
          ld      a,(grpacy)        ;obtém a coord. y
          cp      24                ;é maior do que 23?
          jp      nc,erro_05        ;Sim, volta com erro
          call    lefrase           ;lê a string a inverter
inverte3  di                        ;desabilita as interrupções
          ld      hl,(tabpad)       ;hl=end. da tab. de padrões
          push    hl                ;salva o end. da tab. de
                                    ;padrões
          ld      de,#e0*8          ;calcula o endereço do
          add     hl,de             ;desenho do caractere #e0
          ex      de,hl             ;de=end. do des. do carac. #e0
          ld      c,#e0             ;c=#e0
          ld      b,(ix+#00)        ;b=número de carac. da string
          ld      ix,bufnor+#03     ;ix=endereço da string normal
          ld      iy,bufinv+#03     ;iy=end. da string invertida
          pop     hl                ;recupera hl
loopinv1  push    bc                ;salva bc
          push    hl                ;salva hl
          push    de                ;salva de
          ld      de,#0008          ;bytes para o desenho de cada
                                    ;caractere
          ld      b,(ix+#00)        ;b=caractere a inverter
loopinv2  add     hl,de             ;calcula o endereço desse
          djnz    loopinv2          ;caractere
          pop     de                ;recupera de
          ld      b,#08             ;prepara a modificação
loopinv3  call    rdvram            ;lê o byte original
          cpl                       ;inverte
          ex      de,hl             ;transfere para o novo
          call    wtvram            ;destino
          ex      de,hl             ;
          inc     hl                ;hl=hl+1
```

```
        inc     de              ;de=de+1
        djnz    loopinv3        ;b=b-1
        pop     hl              ;recupera hl
        pop     bc              ;recupera bc
        ld      (iy+#00),c      ;modifica a string
        inc     c               ;c=c+1
        inc     ix              ;aponta para o próximo
                                ;caracter
        inc     iy
        djnz    loopinv1        ;repete até o fim da string
envinvert
        ld      iy,bufinv       ;recupera o ponteiro do
                                ;buffer
        ld      l,(iy+#01)      ;hl=end. de escrita da string
        ld      h,(iy+#02)      ;
        call    setvdpwt        ;ajusta o VDP para escrita
        ld      b,(iy+#00)      ;b=núm. de carac. da string
        ld      hl,bufinv+#03   ;end. da string
        call    esclinha        ;escreve a frase invertida
        ei                      ;habilita as interrupções
        ld      hl,(auxiliar)   ;recupera ptr. do BASIC
        jp      volta           ;volta para o BASIC

tabinver
        defb    #00             ;posição x
        defb    #00             ;posição y
        defb    #00             ;comprimento
        defb    #00             ;flag de restauração

newcls
        ld      b,#02           ;b=núm. de valores
        ld      ix,tabnewcls    ;ix->tab. de valores default
        call    lecomandos      ;
        ld      (auxiliar),hl   ;salva o ponteiro do BASIC
        ld      a,(ix+#00)      ;obtém o primeiro valor
        cp      #02             ;verifica o limite
        jp      nc,erro_05      ;se estiver fora, volta com
                                ;erro
        call    calpadvid       ;calcula os padrões do vídeo
        ld      a,(colunas)     ;obtém o número de colunas
        ld      e,a             ;e=número de colunas
        ld      a,(ix+#01)      ;a=número de rotações
        inc     e
        cp      e               ;rotações>colunas?
        jp      nc,erro_05      ;Sim, volta com erro
```

```
                or      a               ;Zero rotações?
                jp      z,volta         ;Sim, volta sem erro
                ld      a,(ix+#00)      ;a=tipo de rotação
                or      a               ;é zero (esquerda)?
                jp      nz,newcls_0     ;Não, vai para newcls_0


;newcls_1 faz o cls com rotação para a esquerda

newcls_1

                di                      ;desabilita as interrupções
                ld      b,(ix+#01)      ;b=número de rotações
loopcls_11
                push    bc              ;salva contador externo
                ld      hl,(txtnam)     ;hl=end. tabela de nomes
                ld      a,(colunas)     ;a=núm. de colunas
                ld      e,a             ;de=núm. de colunas
                ld      d,#00           ;
                ld      a,(crtcnt)      ;a=número de linhas
                ld      b,a             ;b=número de linhas
loopcls_12
                push    bc              ;salva contador intermediário
                call    setvdprd        ;prepara o VDP para leitura
                ld      a,(colunas)     ;a=núm. de colunas
                push    de              ;salva de
                push    hl              ;salva hl
                ld      hl,bufferlinha  ;hl aponta para o buffer
                ld      b,a             ;b=núm. de colunas
                call    lelinha         ;lê uma linha
                ld      a,#20           ;a=carac. espaço em branco
                ld      (hl),a          ;coloca o espaço na últ. pos.
                pop     hl              ;recupera hl
                pop     de              ;recupera de
                call    setvdpwt        ;prepara o VDP para escrita
                push    de              ;salva de
                push    hl              ;salva hl
                ld      hl,bufferlinha+1 ;hl aponta para o buffer+1
                ld      a,(colunas)     ;a=núm. de colunas
                ld      b,a             ;b=núm. de colunas
                call    esclinha        ;envia a linha já rotada
                pop     hl              ;recupera hl
                pop     de              ;recupera de
                add     hl,de           ;hl aponta para a próx. linha
                pop     bc              ;recupera bc
                djnz    loopcls_12      ;repete para as demais linhas
```

```
            pop     bc                  ;recupera bc
            djnz    loopcls_11          ;repete até terminar todas
                                        ;as rotações
            ei                          ;habilita as interrupções
            ld      hl,(auxiliar)       ;recupera o ponteiro do BASIC
            jp      volta               ;retorna ao BASIC


;newcls_0 faz o cls com rotação para a direita

newcls_0
            di                          ;desabilita as interrupções
            ld      b,(ix+#01)          ;b=número de rotações
loopcls_01
            push    bc                  ;salva contador externo
            ld      hl,(txtnam)         ;hl=end. tabela de nomes
            ld      a,(colunas)         ;a=núm. de colunas
            ld      e,a                 ;de=núm. de colunas
            ld      d,#00               ;
            ld      a,(crtcnt)          ;a=número de linhas
            ld      b,a                 ;b=número de linhas
loopcls_02
            push    bc                  ;salva contador intermediário
            call    setvdprd            ;prepara o VDP para leitura
            ld      a,(colunas)         ;a=núm. de colunas
            push    de                  ;salva de
            push    hl                  ;salva hl
            ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
            ld      b,a                 ;b=núm. de colunas
            call    lelinha             ;lê uma linha
            ld      a,#20               ;a=carac. de espaço em branco
            ld      hl,bufferlinha      ;hl aponta para o buffer
            ld      (hl),a              ;coloca o espaço na prim.
                                        ;posição
            pop     hl                  ;recupera hl
            pop     de                  ;recupera de
            call    setvdpwt            ;prepara o VDP para escrita
            push    de                  ;salva de
            push    hl                  ;salva hl
            ld      hl,bufferlinha      ;hl aponta para o buffer
            ld      a,(colunas)         ;a=núm. de colunas
            ld      b,a                 ;b=núm. de colunas
            call    esclinha            ;envia a linha já rotada
            pop     hl                  ;recupera hl
            pop     de                  ;recupera de
            add     hl,de               ;hl aponta para a próx. linha
```

```
                pop     bc              ;recupera bc
                djnz    loopcls_02      ;repete para as demais linhas
                pop     bc              ;recupera bc
                djnz    loopcls_01      ;repete até terminar todas
                                        ;as rotações
                ei                      ;habilita as interrupções
                ld      hl,(auxiliar)   ;recupera o ponteiro do BASIC
                jp      volta           ;retorna ao BASIC


tabnewcls
                defb    #00             ;tipo de cls
                defb    #00             ;número de rotações


;a rotina calpadvid calcula os padrões do vídeo: a tabela de
;padrões e o número de colunas. Os valores resultantes são
;colocados em tabpad e colunas, respectivamente.

calpadvid
                ex      af,af'          ;salva o registro af
                exx                     ;salva os demais registros
                ld      hl,(txtcgp)     ;obtém a tab. de padrões
                ld      a,(versao)      ;obtém a versão
                or      a               ;do MSX. É MSX1?
                jr      z,calpad_1      ;Sim, vai para calpad_1
                ld      a,(linlen)      ;obtém o tamanho da tela
                cp      41              ;O MSX2 está em 80 colunas
                jr      c,calpad_1      ;Não, vai para calpad_1
                ld      a,80            ;Sim, a=80 colunas
                add     hl,hl           ;calcula novo end. da
                                        ;tabela de padrões
                jr      calpad_2        ;vai para calpad_2
calpad_1        ld      a,40            ;a=40 colunas
calpad_2        ld      (colunas),a     ;salva as colunas
                ld      (tabpad),hl     ;salva o end. da tab. de
                                        ;padrões
                exx                     ;recupera os registros salvos
                ex      af,af'          ;
                ret                     ;retorna


;a rotina lefrase transporta da tela para o buffer bufnor
;a sentença a inverter
```

```
lefrase
            push    af              ;salva os registros
            push    bc              ;modificados por esta
            push    de              ;rotina
            push    hl              ;
            call    calendfra       ;calcula o end. da string
                                    ;na VRAM
            call    setvdprd        ;prepara o VDP para leitura
            ld      b,(ix+#00)      ;obtém o comp. da string
            ld      hl,bufnor+#03   ;hl->início da string
            call    lelinha         ;lê a string
            pop     hl              ;recupera os registros
            pop     de              ;salvos
            pop     bc              ;
            pop     af              ;
            ret                     ;retorna


;a rotina calendfra,baseada nas coordenadas fornecidas, calcula o endereço da sentença a
;inverter na memória VRAM

calendfra
            ld      h,#00           ;calcula o end. da string
            ld      d,h             ;na memória VRAM, baseada
            ld      a,(grpacy)      ;nas coordenadas passadas
            ld      l,a             ;
            or      a               ;a string está na linha 0?
            jr      z,calend1       ;Sim, vai para calend1
            ld      l,h             ;Não, calcula o end. do
            ld      b,a             ;início da linha
            ld      a,(colunas)     ;
            ld      e,a             ;
loopcal_1   add     hl,de           ;
            djnz    loopcal_1       ;
calend1 ld  a,(grpacx)              ;obtém a coord. x
            ld      e,a             ;e=coord. x
            add     hl,de           ;soma ao end. já encontrado
            ld      (bufnor+#01)    ,hl;salva no buffer das strings
            ld      (bufinv+#01)    ,hl;normal e invertida
            ret                     ;retorna


;início da lista com os nomes dos novos comandos e
;endereços de chamada
```

```
lista
                defm    "REVERSE"       ;nome do comando
                defb    #00             ;fim do nome
                defw    inverte         ;end. da rotina
                defm    "CLRSCR"        ;nome do comando
                defb    #00             ;fim do nome
                defw    newcls
endlista        defb    #00             ;fim da lista


;rotinas para o acesso direto à RAM de vídeo

rdvram
                call    setvdprd        ;ajusta o VDP para leitura
                in      a,(#98)         ;lê um byte
                ret                     ;retorna


wtvram
                push    af              ;salva o dado a enviar
                call    setvdpwt        ;ajusta o VDP para escrita
                pop     af              ;recupera o dado a enviar
                out     (#98),a         ;envia
                ret                     ;retorna

setvdprd
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,rdvram1       ;Sim, vai para rdvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a         ;
rdvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde será
                and     #3f             ;lido o dado
                out     (#99),a         ;
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                ret


setvdpwt
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
```

```
                jr      z,wtvram1        ;Sim, vai para wtvram1
                xor     a                ;Não, inicializa o VDP
                out     (#99),a          ;do MSX2
                ld      a,#8e
                out     (#99),a
wtvram1         ld      a,l              ;informa ao
                out     (#99),a          ;VDP o endereço na
                ld      a,h              ;VRAM onde o
                and     #3f              ;dado será
                or      #40              ;gravado
                out     (#99),a          ;
                ex      (sp),hl          ;demora para
                ex      (sp),hl          ;sincronização
                ret                      ;retorna


;a rotina lelinha transporta uma linha da VRAM para a RAM

lelinha
                in      a,(#98)          ;lê o carac. da VRAM
                ld      (hl),a           ;salva-o no buffer
                inc     hl               ;incrementa o ponteiro
                djnz    lelinha          ;prepara a próxima leitura
                ret                      ;retorna


;a rotina esclinha transporta uma linha da RAM para a VRAM

esclinha
                ld      a,(hl)           ;lê o carac. do buffer
                out     (#98),a          ;escreve-o na VRAM
                inc     hl               ;incrementa o ponteiro
                djnz    esclinha         ;prepara a próxima escrita
                ret                      ;retorna


;área das variáveis usadas no programa

bufnor
                defb    #00              ;tamanho da string
                defw    #0000            ;posição de escrita
                defs    33               ;string

bufinv
                defb    #00              ;tamanho da string
                defw    #0000            ;posição de escrita
```

```
                defs    33              ;string
inversao defb   #00                     ;flag de campo já invertido
compstr defb    #00                     ;comprimento da string
colunas defb    #00                     ;número de colunas na tela
tabpad  defw    #0000                   ;endereço da tab. de padrões
auxiliar defw   #0000                   ;ponteiro do BASIC
bufferlinha     defs    81              ;buffer de linha da tela
fim             equ     $
```

Listagem em linhas data do código-objeto do programa que implementa o comando CLRSCR:

```
10 FOR A%=&HD000 TO &HD3D4
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG23.BIN",&HD000,&HD3D4
100 DATA F3,DB,A8,47,E6,F0,0F,0F,0F,0F,B0,D3,A8,E6,03,32
110 DATA 2E,D0,78,21,2D,D0,11,B1,FF,01,05,00,ED,B0,21,32
120 DATA D0,11,00,40,01,A2,03,ED,B0,D3,A8,FB,C9,F7,00,00
130 DATA 40,C9,F5,C5,D5,E5,7B,FE,02,C2,20,40,2A,AF,F6,7E
140 DATA B7,C2,16,40,23,23,23,23,DD,21,66,46,CD,AD,40,CD
150 DATA 25,40,E1,D1,C1,F1,C9,22,4E,43,21,9F,42,01,13,00
160 DATA ED,5B,4E,43,1A,ED,B1,E2,48,40,13,1A,BE,C2,2E,40
170 DATA 23,7E,FE,00,CA,49,40,C3,38,40,C9,23,4E,23,46,F1
180 DATA C5,ED,53,4E,43,EB,C9,DD,E5,DD,E5,DD,21,66,46,CD
190 DATA AD,40,DD,E1,28,1A,FE,2C,28,12,2B,DD,E5,C5,DD,21
200 DATA 0E,52,CD,AD,40,C1,DD,E1,7B,DD,77,00,DD,23,10,D9
210 DATA DD,E1,C9,3E,05,32,14,F4,C3,89,40,E1,D1,C1,F1,3A
220 DATA 14,F4,5F,C9,2B,AF,32,14,F4,D1,D1,C1,F1,D1,C1,F1
230 DATA DD,E1,DD,E1,F5,C5,D5,DD,21,66,46,C3,AD,40,C9,CD
240 DATA 59,01,C9,06,04,DD,21,83,41,CD,55,40,DD,7E,00,32
250 DATA B7,FC,DD,7E,01,32,B9,FC,DD,7E,02,32,4A,43,FE,21
260 DATA D2,81,40,22,4E,43,DD,7E,03,B7,28,16,3A,49,43,B7
270 DATA 28,10,2A,02,43,CD,D9,42,21,04,43,3A,01,43,47,CD
280 DATA FA,42,3E,FF,32,49,43,2A,4E,43,3A,4A,43,B7,CA,92
290 DATA 40,DD,21,01,43,FD,21,25,43,DD,77,00,FD,77,00,CD
300 DATA 46,42,3A,B7,FC,5F,3A,4B,43,BB,DA,81,40,3A,B9,FC
310 DATA FE,18,D2,81,40,CD,68,42,F3,2A,4C,43,E5,11,00,07
```

```
320 DATA 19,EB,0E,E0,DD,46,00,DD,21,04,43,FD,21,28,43,E1
330 DATA C5,E5,D5,11,08,00,DD,46,00,19,10,FD,D1,06,08,CD
340 DATA B3,42,2F,EB,CD,B9,42,EB,23,13,10,F3,E1,C1,FD,71
350 DATA 00,0C,DD,23,FD,23,10,D8,FD,21,25,43,FD,6E,01,FD
360 DATA 66,02,CD,D9,42,FD,46,00,21,28,43,CD,FA,42,FB,2A
370 DATA 4E,43,C3,92,40,00,00,00,00,06,02,DD,21,44,42,CD
380 DATA 55,40,22,4E,43,DD,7E,00,FE,02,D2,81,40,CD,46,42
390 DATA 3A,4B,43,5F,DD,7E,01,1C,BB,D2,81,40,B7,CA,92,40
400 DATA DD,7E,00,B7,C2,FB,41,F3,DD,46,01,C5,2A,B3,F3,3A
410 DATA 4B,43,5F,16,00,3A,B1,F3,47,C5,CD,C1,42,3A,4B,43
420 DATA D5,E5,21,50,43,47,CD,F3,42,3E,20,77,E1,D1,CD,D9
430 DATA 42,D5,E5,21,51,43,3A,4B,43,47,CD,FA,42,E1,D1,19
440 DATA C1,10,D6,C1,10,C5,FB,2A,4E,43,C3,92,40,F3,DD,46
450 DATA 01,C5,2A,B3,F3,3A,4B,43,5F,16,00,3A,B1,F3,47,C5
460 DATA CD,C1,42,3A,4B,43,D5,E5,21,51,43,47,CD,F3,42,3E
470 DATA 20,21,50,43,77,E1,D1,CD,D9,42,D5,E5,21,50,43,3A
480 DATA 4B,43,47,CD,FA,42,E1,D1,19,C1,10,D3,C1,10,C2,FB
490 DATA 2A,4E,43,C3,92,40,00,00,08,D9,2A,B7,F3,3A,2D,00
500 DATA B7,28,0C,3A,B0,F3,FE,29,38,05,3E,50,29,18,02,3E
510 DATA 28,32,4B,43,22,4C,43,D9,08,C9,F5,C5,D5,E5,CD,80
520 DATA 42,CD,C1,42,DD,46,00,21,04,43,CD,F3,42,E1,D1,C1
530 DATA F1,C9,26,00,54,3A,B9,FC,6F,B7,28,09,6C,47,3A,4B
540 DATA 43,5F,19,10,FD,3A,B7,FC,5F,19,22,02,43,22,26,43
550 DATA C9,52,45,56,45,52,53,45,00,B1,40,43,4C,52,53,43
560 DATA 52,00,87,41,00,CD,C1,42,DB,98,C9,F5,CD,D9,42,F1
570 DATA D3,98,C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99
580 DATA 7D,D3,99,7C,E6,3F,D3,99,E3,E3,C9,3A,2D,00,B7,28
590 DATA 07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6,40
600 DATA D3,99,E3,E3,C9,DB,98,77,23,10,FA,C9,7E,D3,98,23
610 DATA 10,FA,C9,00,00,00,00,00,00,00,00,00,00,00,00,00
620 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
630 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
640 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
650 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
660 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
670 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
680 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
690 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
700 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
710 DATA 00,00,00,23,39
```

Antes de passar para o próximo capítulo, gostaria de colocar novamente as duas exigências no uso de comandos implementados pelo desvio do vetor de erro. São elas:

1.Nunca começar o nome de um novo comando com um token. Se esta exigência não for cumprida, o interpretador BASIC irá gerar uma mensagem de erro de sintaxe, mesmo que o vetor

de erro esteja desviado. O comando **SAVESCR** é um exemplo típico, já que a palavra **SAVE** é um token do interpretador BASIC. Para contornar este problema, sugiro a inversão para **SCRSAVE**.

2.Nunca use um comando resultante do desvio do vetor de erro após o comando **THEN**. Dê preferência ao uso do comando **GOSUB**, que eliminará esse problema e, por tabela, promoverá uma maior estruturação do seu programa em BASIC.

Resta agora apresentar um pequeno programa que nos permita conhecer os bytes/caracteres que formam um determinado comando criado por nós. Particularmente, utilizo um pequeno programa em BASIC para tal tarefa. Vamos, então, à listagem desse programa:

Listagem do programa em BASIC para descobrir os bytes/caracteres que formam um novo comando:

```
100 GOTO 110:SCRSAVE
110 A%=&H800B
120 CLS:KEYOFF
130 S$="FORMAÇÃO DO NOVO COMANDO"
140 IF PEEK (&HF3B0)> 40 THEN CP%=80 ELSE CP%=40
150 LOCATE (CP%-LEN(S$))/2,7:PRINT S$
160 LOCATE 0,9:PRINT "Bytes      :"
170 LOCATE 0,11:PRINT "Caracteres :"
180 CT%=0
190 IF PEEK(A%)=0 THEN END
200 LOCATE 13+CT%,9:PRINT RIGHT$("00"+HEX$(PEEK(A%)),2)
210 LOCATE 13+CT%,11:PRINT CHR$(PEEK(A%))
220 A%=A%+1:CT%=CT%+3
230 GOTO 190
```

Antes de executar o programa acima, certifique-se de ter entrado com a listagem exatamente como apresentada, já que este é um passo imprescindível para a execução correta do programa. Observe que você deve entrar com o comando a ser criado na linha **100** logo após a sentença **GOTO 110:**. Na listagem acima, o exemplo usado foi o do comando **SCRSAVE**. O uso deste programa ficará claro nos exemplos dos capítulos seguintes.

Com a listagem acima, terminamos a apresentação e estudo dos hooks no MSX. No próximo capítulo, vamos nos dedicar ao estudo do controle dos dispositivos de seleção, mais especificamente das teclas do cursor e dos joysticks.

BIBLIOGRAFIA RECOMENDADA

INTRODUÇÃO À LINGUAGEM DE MÁQUINA PARA MSX - Eduardo Alberto Barbosa - Rio de Janeiro - CIÊNCIA MODERNA COMPUTAÇÃO LTDA.

Capítulo 3

OS DISPOSITIVOS DE SELEÇÃO

Como já afirmei no primeiro capítulo, é cada vez mais freqüente o uso de uma interface amigável como meio de programação/utilização do microcomputador. Ocorre que, embora a última palavra esteja na interface gráfica, podemos fazer maravilhas unindo a simplicidade da tela de texto aos princípios de uma interface amigável. O que proponho é usar os dispositivos para controle de direção (teclas do cursor ou joystick) para fazer a seleção apropriada num menu. Vou exemplificar, para tornar mais clara a minha proposta. Você já deve ter observado que os programas mais antigos fazem as seleções com menus do tipo:

1. Carregar o processador de textos

2. Carregar a planilha

3. Carregar o programa gráfico

4. Sair

Escolha uma das opções acima (1-4):_

onde você deverá pressionar um número no teclado para executar uma determinada opção. Este método foi a forma mais amigável de interação com o usuário durante alguns anos, até o surgimento de uma nova interface mais versátil e clara: a interface que emprega a inversão de vídeo para a opção atual. A principal vantagem deste último método está em permitir uma visualização mais rápida e nítida da escolha atual, em contraste com o método anterior, que não permite tal recurso. A outra vantagem do método por inversão se situa na utilização somente das teclas do cursor ou joystick para selecionar as diversas opções, ao contrário do método anterior, que exige a utilização do teclado. Como você já deve estar prevendo, a técnica de escolha por inversão de vídeo é muito mais amigável e direta, tornando o seu uso praticamente obrigatório. "Bem, Eduardo, é realmente tudo muito bonito, mas como usar tal técnica no MSX?" Eu respondo com outra pergunta: Que tal criar um comando específico para essa tarefa? Você já deve ter percebido que a chave está na inversão de vídeo, mas este é um assunto que já dominamos desde o primeiro capítulo. Partindo deste princípio, proponho a criação do seguinte comando:

MENU *X1,Y1,X2,Y2,Comp,Des,VInt*
onde

X1 representa a posição **x** do primeiro caractere da primeira opção do menu;

Y1 representa a posição **y** do primeiro caractere da primeira opção do menu;

X2 representa a posição **x** do primeiro caractere da última opção do menu;

Y2 representa a posição **y** do primeiro caractere da última opção do menu;

Comp representa o comprimento (em caracteres) da maior opção do menu;

Des representa o deslocamento (vertical ou horizontal) de uma opção para outra, e;

VInt representa a variável **inteira** que receberá o valor numérico da seleção feita.

O COMANDO WINDOW

O comando **MENU** será implementado utilizando a nossa estrutura para o desvio do vetor de erro. Uma vez implementado, o comando **MENU** facilitará em muito o projeto de programas mais amigáveis em BASIC. Porém, antes de partir para as explicações e listagens do comando acima, gostaria de implementar um outro comando, o comando **WINDOW**, que, como o nome já afirma, trata-se de um comando para desenhar janelas na tela de texto. Tendo em vista que a rotina para o desenho de janelas já está pronta desde o primeiro capítulo, só nos resta programar a leitura dos valores correspondentes às coordenadas e ao tipo de apresentação. Para tanto, vamos criar um novo comando com a seguinte sintaxe:

WINDOW *X1,Y1,X2,Y2,Tipo*

onde

X1 representa a posição **x** do canto superior esquerdo da janela a ser criada;

Y1 representa a posição **y** do canto superior esquerdo da janela a ser criada;

X2 representa a posição **x** do canto inferior direito da janela a ser criada;

Y2 representa a posição **y** do canto inferior direito da janela a ser criada, e;

Tipo representa o tipo da apresentação: com ou sem explosão.

Embora os comandos acima utilizem uma entrada de coordenadas que não é padrão, pois o normal seria a entrada de pares de coordenadas entre parênteses tal como acontece no comando **LINE**, por exemplo, vamos utilizar este método não-padrão por duas razões:

1.O interpretador BASIC não aceitaria a entrada de um comando como

WINDOW (30,0)-(50,10),1

já que interpretaria os argumentos **(30,0)** como referência a uma matriz chamada **window**, o que resultaria num erro de subscrito fora de faixa.

2.No Capítulo 2, desenvolvemos uma rotina chamada **lecomandos**, que interpreta os valores passados como argumentos e separados por vírgulas, então, por que reinventar a roda?

Feitos os esclarecimentos, vamos às listagens do programa que implementa os comandos do Capítulo 2 acrescidos do comando **WINDOW**.

Listagem em assembly Z-80 do código-fonte do programa que implementa o comando WINDOW:

```
;programa que implementa o comando WINDOW
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG24.BIN=PROG24.GEN
;
;onde PROG24.GEN é o nome do arquivo-texto com esta
;listagem

versao      equu    #002d
```

```
                defw    fim-erro+rotina+#01
                                        ;um arquivo
                defw    inicio          ;.BIN


                org     #d000

inicio
                di                      ;desabilita as interrupções
                in      a,(#a8)         ;lê a atual configuração dos
                ld      b,a             ;slots e prepara para ativar
                and     #f0             ;a página 1 do slot da RAM
                rrca                    ;
                rrca                    ;
                rrca                    ;
                rrca                    ;
                or      b               ;
                out     (#a8),a         ;ativa as páginas 1, 2 e 3 em
                and     #03             ;RAM e descobre o slot onde
                ld      (slot),a        ;se encontram os 64Kb de RAM
                ld      a,b             ;a=config. inicial dos slots
                ld      hl,novohook     ;promove o desvio do hook
                ld      de,herro        ;dos erros
                ld      bc,0005         ;
                ldir                    ;
                ld      hl,rotina       ;transfere a rotina para a
                ld      de,erro         ;página 1 da RAM
                ld      bc,fim-erro+#01 ;
                ldir                    ;
                out     (#a8),a         ;habilita a config. original
                ei                      ;habilita as interrupções
                ret                     ;retorna ao BASIC


novohook
                defb    #f7             ;RST #30
slot            defb    #00             ;identificação do slot
                defw    erro            ;endereço de desvio
                defb    #c9             ;RET



rotina
                org     #4000

erro
                push    af              ;salva os registros afetados
```

```
                push    bc              ;
                push    de              ;
                push    hl              ;
examerro        ld      a,e             ;a=código do erro
                cp      #02             ;é erro de sintaxe?
                jp      nz,aborta       ;Não, volta ao BASIC
pegachar        ld      hl,(savtxt)     ;Sim, obtém a posição do
                                        ;ponteiro do BASIC
                ld      a,(hl)          ;é final de linha?
                or      a               ;
                jp      nz,proxchar     ;Não, obtém o próximo carac.
pula            inc     hl              ;Sim, pula as 4 posições
                inc     hl              ;referentes ao número da
                inc     hl              ;linha e ao end. da próxima
                inc     hl              ;linha
proxchar        ld      ix,chrgtr       ;obtém o primeiro caractere
                call    chamabasic      ;do novo comando
compara         call    compstring      ;compara com a lista de novos
                                        ;comandos
aborta          pop     hl              ;recupera os registros salvos
                pop     de              ;
                pop     bc              ;
                pop     af              ;
                ret                     ;retorna ao BASIC


;a rotina compstring é a responsável pela localização da
;rotina do novo comando

compstring
                ld      (auxiliar),hl   ;salva o pont. do BASIC
                ld      hl,lista        ;hl aponta p/ lista de
                                        ;comandos
                ld      bc,endlista-lista
                                        ;bc=tamanho da lista
compstr1        ld      de,(auxiliar)   ;de aponta p/ comando em
                                        ;BASIC
                ld      a,(de)          ;obtém o primeiro byte
                cpir                    ;tenta encontrá-lo na
                                        ;lista
                jp      po,naoachei     ;se bc=0 -> não achou
compstr2        inc     de              ;achou o primeiro e
                                        ;parte
                ld      a,(de)          ;para a comparação dos
                cp      (hl)            ;demais caracteres. Vai
                                        ;para
                jp      nz,compstr1     ;compstr1 se achar um
```

```
                                            ;byte diferente.
                inc     hl                  ;Caso contrário,
                ld      a,(hl)              ;continua até o fim do
                                            ;comando.
                cp      #00                 ;Se chegou ao fim e,
                jp      z,achei             ;então, achou o comando
                                            ;na lista.
                jp      compstr2            ;Caso contrário, compara
naoachei        ret                         ;o próximo byte.
achei           inc     hl                  ;Se achou, obtém o
                                            ;endereço
                ld      c,(hl)              ;de entrada da rotina,
                inc     hl                  ;colocando-o em BC
                ld      b,(hl)              ;
                pop     af                  ;retira da pilha o
                                            ;endereço de
                push    bc                  ;retorno para a rotina
                                            ;erro e
                                            ;coloca na pilha o
                                            ;endereço de
                                            ;entrada da rotina do
                                            ;novo comando
                ld      (auxiliar),de       ;guarda o ponteiro
                                            ;do BASIC
                ex      de,hl               ;transfere o ponteiro
                                            ;para HL
                ret                         ;usa RET para dar um
                                            ;jump
                                            ;para o endereço contido
                                            ;BC que foi salvo na
                                            ;pilha


;a rotina lecomandos é a responsável pela interpretação dos
;valores que se seguem ao nome do novo comando. Os valores
;devem estar separados por vírgulas. O par IX na entrada
;aponta para uma tabela com valores default e o registro B
;contém o número de valores a interpretar

lecomandos
                push    ix                  ;salva o par IX
lecoman_1       push    ix                  ;
                ld      ix,chrgtr           ;obtém um valor
                call    chamabasic
                pop     ix                  ;recupera o ponteiro IX
                jr      z,lecoman_4         ;Se não houver valor, vai para
                                            ;lecoman_4
```

```
            cp      ","             ;é vírgula?
            jr      z,lecoman_3     ;Sim, assume o default
            dec     hl              ;Não, obtém o valor
            push    ix              salva o ponteiro
            push    bc              ;salva o contador
            ld      ix,evlexp       ;obtém o valor
            call    chamabasic      ;
            pop     bc              ;recupera o valor
            pop     ix              ;recupera o ponteiro
            ld      a,e             ;a=valor interpretado
            ld      (ix+#00),a      ;coloca-o na tabela
lecoman_3   inc     ix              ;incrementa o ponteiro
            djnz    lecoman_1       ;repete para os demais
                                    ;valores
lecoman_4   pop     ix              ;recupera o ponteiro salvo
            ret                     ;retorna


;a rotina erro_05 emite a mensagem de chamada ilegal

erro_05
            ld      a,#05           ;a=código do erro
            ld      (errflg),a      ;salva o erro em errflg
            jp      voltaerro       ;vai para voltaerro


;a rotina voltaerro substitui o erro original por um outro

voltaerro
            pop     hl              ;recupera os pares
            pop     de              ;salvos no início da
            pop     bc              ;rotina erro
            pop     af              ;
            ld      a,(errflg)      ;a=erro a ser emitido
            ld      e,a             ;e=erro a ser emitido
            ret                     ;volta para a rotina
                                    ;de manipulação de erros
                                    ;do interpretador BASIC


;a rotina volta promove a volta ao BASIC sem que haja
;interrupção no processamento, ou seja, sem que o sistema
;detecte o erro ocorrido

volta
            dec     hl              ;hl aponta para o final
                                    ;do último comando
```

```
            xor     a               ;modifica para nenhum erro
            ld      (errflg),a      ;ocorrido
            pop     de              ;recupera os registros
            pop     de              ;salvos no início da rotina
            pop     bc              ;erro. Note que hl não é
            pop     af              ;recuperado.
            pop     de              ;obtém o end. de retorno da
                                    ;nossa rotina para a rotina
                                    ;de chaveamento de slots
            pop     bc              ;obtém o end. de retorno da
                                    ;rotina de chaveamento de
                                    ;slots para RST #30
            pop     af              ;obtém o end. de retorno de
                                    ;RST #30 para o hook
            pop     ix              ;obtém/destrói o end. de
                                    ;retorno do hook para a
                                    ;rotina
                                    ;de erro do interpretador
                                    ;BASIC
            pop     ix              ;obtém/destrói o end. de
                                    ;retorno
                                    ;da rotina de erro para o
                                    ;interpretador BASIC
            push    af              ;repõe na pilha os end. de
            push    bc              ;retorno salvos nestes
            push    de              ;registros
            ld      ix,chrgtr       ;faz com que hl aponte
            jp      chamabasic      ;para o próximo comando
            ret                     ;e retorna ao BASIC


chamabasic
            call    calbas          ;chama a rotina calbas
            ret                     ;retorna

inverte
            ld      b,#04           ;b=núm. de valores
            ld      ix,tabinver     ;ix=tabela default
            call    lecomandos      ;lê os 4 valores
            ld      a,(ix+#00)      ;obtém coord. x
            ld      (grpacx),a      ;salva em grpacx
            ld      a,(ix+#01)      ;obtém coord. y
            ld      (grpacy),a      ;salva em grpacy
            ld      a,(ix+#02)      ;obtém o comp. da string
            ld      (compstr),a     ;salva em compstr
            cp      33              ;é >= 33?
            jp      nc,erro_05      ;Sim, chamada ilegal
```

```
          ld      (auxiliar),hl     ;salva ptr. do BASIC
          ld      a,(ix+#03)        ;a=flag de recuperação
          or      a                 ;está setada?
          jr      z,inverte0        ;Não, vai para inverte0
          ld      a,(inversao)      ;Sim, já foi feita alguma
          or      a                 ;inversão?
          jr      z,inverte0        ;Não, vai para inverte0
          ld      hl,(bufnor+#01)   ;Sim, obtém a pos. da string
          call    setvdpwt          ;antiga e prepara o VDP
          ld      hl,bufnor+#03     ;hl->início da string antiga
          ld      a,(bufnor)        ;a=comp. da string antiga
          ld      b,a               ;a=comp. da string antiga
          call    esclinha          ;escreve a string antiga
                                    ;no modo normal
inverte0  ld      a,#ff             ;
          ld      (inversao),a      ;sinaliza uma inversão
          ld      hl,(auxiliar)     ;hl=ponteiro do BASIC
          ld      a,(compstr)       ;a=comp. da string
          or      a                 ;comprimento=0?
          jp      z,volta           ;Sim, volta com erro
          ld      ix,bufnor         ;Não, ajusta os buffers
          ld      iy,bufinv         ;para a string normal e
          ld      (ix+#00),a        ;para a string invertida
          ld      (iy+#00),a        ;com o comprimento da
          call    calpadvid         ;calcula os padrões do vídeo
          ld      a,(grpacx)        ;obtém a coord. x
          ld      e,a               ;coloca em e
          ld      a,(colunas)       ;obtém o número de colunas
          cp      e                 ;compara com o valor lido
          jp      c,erro_05         ;Se valor lido > colunas
                                    ;volta com erro
          ld      a,(grpacy)        ;obtém a coord. y
          cp      24                ;é maior do que 23?
          jp      nc,erro_05        ;Sim, volta com erro
          call    lefrase           ;lê a string a inverter
inverte3  di                        ;desabilita as interrupções
          ld      hl,(tabpad)       ;hl=end. da tab. de padrões
          push    hl                ;salva o end. da tab.
                                    ;de padrões
          ld      de,#e0*8          ;calcula o endereço do
          add     hl,de             ;desenho do caractere #e0
          ex      de,hl             ;de=end. do desenho do
                                    ;caractere #e0
          ld      c,#e0             ;c=#e0
          ld      b,(ix+#00)        ;b=número de carac. da string
          ld      ix,bufnor+#03     ;ix=end. da string normal
          ld      iy,bufinv+#03     ;iy=end. da string invertida
```

```
               pop     hl              ;recupera hl
loopinv1       push    bc              ;salva bc
               push    hl              ;salva hl
               push    de              ;salva de
               ld      de,#0008        ;bytes para o desenho
                                       ;de cada caractere
               ld      b,(ix+#00)      ;b=caractere a inverter
loopinv2       add     hl,de           ;calcula o endereço desse
               djnz    loopinv2        ;caractere
               pop     de              ;recupera de
               ld      b,#08           ;prepara a modificação
loopinv3       call    rdvram          ;lê o byte original
               cpl                     ;inverte
               ex      de,hl           ;transfere para o novo
               call    wtvram          ;destino
               ex      de,hl           ;
               inc     hl              ;hl=hl+1
               inc     de              ;de=de+1
               djnz    loopinv3        ;b=b-1
               pop     hl              ;recupera hl
               pop     bc              ;recupera bc
               ld      (iy+#00),c      ;modifica a string
               inc     c               ;c=c+1
               inc     ix              ;aponta para o próximo carac.
               inc     iy
               djnz    loopinv1        ;repete até o fim da string
envinvert
               ld      iy,bufinv       ;recupera o ptr. do buffer
               ld      l,(iy+#01)      ;hl=end. de escrita da string
               ld      h,(iy+#02)      ;
               call    setvdpwt        ;ajusta o VDP para escrita
               ld      b,(iy+#00)      ;b=núm. de carac. da string
               ld      hl,bufinv+#03   ;end. da string
               call    esclinha        ;escreve a frase invertida
               ei                      ;habilita as interrupções
               ld      hl,(auxiliar)   ;recupera ptr. do BASIC
               jp      volta           ;volta para o BASIC


tabinver

               defb    #00             ;posição x
               defb    #00             ;posição y
               defb    #00             ;comprimento
               defb    #00             ;flag de restauração


newcls
```

```
            ld      b,#02           ;b=núm. de valores
            ld      ix,tabnewcls    ;ix->tab. de valores default
            call    lecomandos      ;
            ld      (auxiliar),hl   ;salva o ponteiro do BASIC
            ld      a,(ix+#00)      ;obtém o primeiro valor
            cp      #02             ;verifica o limite
            jp      nc,erro_05      ;se estiver fora, volta com
                                     erro
            call    calpadvid       ;calcula os padrões do vídeo
            ld      a,(colunas)     ;obtém o número de colunas
            ld      e,a             ;e=número de colunas
            ld      a,(ix+#01)      ;a=número de rotações
            inc     e
            cp      e               ;rotações>colunas?
            jp      nc,erro_05      ;Sim, volta com erro
            or      a               ;Zero rotações?
            jp      z,volta         ;Sim, volta sem erro
            ld      a,(ix+#00)      ;a=tipo de rotação
            or      a               ;é zero (esquerda)?
            jp      nz,newcls_0     ;Não, vai para newcls_0


;newcls_1 faz o cls com rotação para a esquerda

newcls_1

            di                      ;desabilita as interrupções
            ld      b,(ix+#01)      ;b=número de rotações
loopcls_11  push    bc              ;salva contador externo
            ld      hl,(txtnam)     ;hl=end. tabela de nomes
            ld      a,(colunas)     ;a=núm. de colunas
            ld      e,a             ;de=núm. de colunas
            ld      d,#00           ;
            ld      a,(crtcnt)      ;a=número de linhas
            ld      b,a             ;b=número de linhas
loopcls_12
            push    bc              ;salva contador intermediário
            call    setvdprd        ;prepara o VDP para leitura
            ld      a,(colunas)     ;a=núm. de colunas
            push    de              ;salva de
            push    hl              ;salva hl
            ld      hl,bufferlinha  ;hl aponta para o buffer
            ld      b,a             ;b=núm. de colunas
            call    lelinha         ;lê uma linha
            ld      a,#20           ;a=carac. espaço em branco
            ld      (hl),a          ;coloca o espaço na últ. pos.
            pop     hl              ;recupera hl
```

```
             pop     de                  ;recupera de
             call    setvdpwt            ;prepara o VDP para escrita
             push    de                  ;salva de
             push    hl                  ;salva hl
             ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
             ld      a,(colunas)         ;a=núm. de colunas
             ld      b,a                 ;b=núm. de colunas
             call    esclinha            ;envia a linha já rotada
             pop     hl                  ;recupera hl
             pop     de                  ;recupera de
             add     hl,de               ;hl aponta para a próx. linha
             pop     bc                  ;recupera bc
             djnz    loopcls_12          ;repete para as demais linhas
             pop     bc                  ;recupera bc
             djnz    loopcls_11          ;repete até terminar todas
                                         ;as rotações
             ei                          ;habilita as interrupções
             ld      hl,(auxiliar)       ;recupera o ponteiro do BASIC
             jp      volta               ;retorna ao BASIC

;newcls_0 faz o cls com rotação para a direita

newcls_0
             di                          ;desabilita as interrupções
             ld      b,(ix+#01)          ;b=número de rotações
loopcls_01
             push    bc                  ;salva contador externo
             ld      hl,(txtnam)         ;hl=end. tabela de nomes
             ld      a,(colunas)         ;a=núm. de colunas
             ld      e,a                 ;de=núm. de colunas
             ld      d,#00               ;
             ld      a,(crtcnt)          ;a=número de linhas
             ld      b,a                 ;b=número de linhas
loopcls_02
             push    bc                  ;salva contador intermediário
             call    setvdprd            ;prepara o VDP para leitura
             ld      a,(colunas)         ;a=núm. de colunas
             push    de                  ;salva de
             push    hl                  ;salva hl
             ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
             ld      b,a                 ;b=núm. de colunas
             call    lelinha             ;lê uma linha
             ld      a,#20               ;a=carac. espaço em branco
             ld      hl,bufferlinha      ;hl aponta para o buffer
             ld      (hl),a              ;coloca o espaço na prim.
                                         ;posição
```

```
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        call    setvdpwt        ;prepara o VDP para escrita
        push    de              ;salva de
        push    hl              ;salva hl
        ld      hl,bufferlinha  ;hl aponta para o buffer
        ld      a,(colunas)     ;a=núm. de colunas
        ld      b,a             ;b=núm. de colunas
        call    esclinha        ;envia a linha já rotada
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        add     hl,de           ;hl aponta para a próx. linha
        pop     bc              ;recupera bc
        djnz    loopcls_02      ;repete para as demais linhas
        pop     bc              ;recupera bc
        djnz    loopcls_01      ;repete até terminar todas
                                ;as rotações
        ei                      ;habilita as interrupções
        ld      hl,(auxiliar)   ;recupera o ponteiro do BASIC
        jp      volta           ;retorna ao BASIC

tabnewcls
        defb    #00             ;tipo de cls
        defb    #00             ;número de rotações

window
        ld      ix,tabwindow    ;ix aponta p/ tabela
        ld      b,#05           ;b=núm. de comandos
        call    lecomandos      ;lê as coordenadas
        ld      (auxiliar),hl   ;salva o ptr. do BASIC
        call    calpadvid       ;calcula os parâmetros
                                ;do vídeo
        ld      ix,tabwindow    ;ix aponta p/ tabela
        ld      h,(ix+#00)      ;h=x1
        ld      l,(ix+#01)      ;l=y1
        ld      d,(ix+#02)      ;d=x2
        ld      e,(ix+#03)      ;e=y2
        ld      a,h             ;testa se x2>x1
        cp      d               ;
        jp      nc,erro_05      ;x2<=x1->erro
        ld      a,l             ;testa se y2>y1
        cp      e               ;
        jp      nc,erro_05      ;y2<=y1->erro
        ld      a,e             ;a=y2
        cp      24              ;y2>=24?
```

```
                jp      nc,erro_05      ;Sim, volta com erro
                ld      a,(colunas)     ;a=colunas na tela
                dec     a
                cp      d               ;x2>=colunas?
                jp      c,erro_05       ;Sim, volta com erro
                ld      a,(ix+#04)      ;a=tipo de apresentação
                or      a               ;é zero?
                jp      nz,zoom         ;Não, vai para zoom
                call    janela          ;Sim, chama janela

windvolta
                ld      hl,(auxiliar)   ;hl=ponteiro do BASIC
                jp      volta           ;volta para o BASIC


;A rotina zoom cria o efeito de explosão da janela

zoom
                xor     a               ;zera o acumulador
                ld      (bufcol),a      ;zera o buffer da coluna
                ld      (buflin),a      ;zera o buffer da linha
                ld      (coror1),hl     ;salva as coordenadas x1,y1
                ld      (coror2),de     ;salva as coordenadas x2,y2
                ld      a,h             ;carrega a com h (x1)
                add     a,d             ;soma com d (x2) e divide
                srl     a               ;por dois para achar Xmedio
                dec     a               ;decrementa Xmedio
                ld      h,a             ;carrega h (x1) com Xmedio
                inc     a               ;incrementa o ponto médio
                                        ;para que
                inc     a               ;x2=x1+1, onde x1=Xmedio-1
                ld      d,a             ;carrega d com x2
                ld      a,l             ;a=y1
                add     a,e             ;soma com e (y2) e divide
                srl     a               ;por dois para achar Ymedio
                dec     a               ;decrementa Ymedio
                ld      l,a             ;carrega l (y1) com Ymedio
                inc     a               ;incrementa o ponto médio
                                        ;para que
                inc     a               ;y2=y1+1, onde y1=Ymedio-1
                ld      e,a             ;carrega e com y2
                call    janela          ;desenha a primeira janela
looptest        call    coluna          ;verifica em rel. à col.
                                        ;limite
                call    linha           ;verifica em rel. à linha
                                        ;limite
                call    janela          ;desenha a janela
```

```
                call    espjanela       ;retardo para tornar o efeito
                                        ;visível
                call    teste2          ;verifica se já chegou à
                                        ;janela final
                jr      nz,looptest     ;Não, continua com a explosão
                ld      hl,(coror1)     ;imprime a janela final
                ld      de,(coror2)     ;
                call    janela          ;
                jp      windvolta       ;retorna ao BASIC

coluna          ld      a,(coror1+#01)  ;a=coord. x1 original
                cp      h               ;já foi atingida?
                jr      nc,fimcol       ;Sim, vai para fimcol
                dec     h               ;Não, x1=x1-1
                inc     d               ;x2=x2+1
                ret                     ;retorna
fimcol          ld      a,#01           ;indica que a coluna-limite
                ld      (bufcol),a      ;foi atingida
                ret                     ;retorna

linha           ld      a,(coror1)      ;a=coord. y1 original
                cp      l               ;já foi atingida?
                jr      nc,fimlin       ;Sim, vai para fimlin
                dec     l               ;Não, y1=y1-1
                inc     e               ;y2=y2+1
                ret                     ;retorna
fimlin          ld      a,#01           ;indica que a linha-limite
                ld      (buflin),a      ;foi atingida
                ret                     ;retorna

espjanela       push    af              ;salva os registros
                push    hl              ;afetados
                ld      hl,#1000        ;altere este valor para mudar
espjanela1
                dec     hl              ;retardo. Decrementa hl
                ld      a,h             ;verifica se chegou
                or      l               ;ao fim
                jr      nz,espjanela1   ;Não, volta para espjanela1
                pop     hl              ;recupera os registros
                pop     af              ;
                ret                     ;e retorna

teste2          ld      a,(bufcol)      ;verifica se a coluna-
                cp      #01             ;limite foi atingida
                jr      z,contes        ;Sim, vai para contes
                ret                     ;Não, retorna
```

```
contes          ld      a,(buflin)      ;verifica se a linha-
                cp      #01             ;limite foi atingida
                ret                     ;retorna. Flag Z será o
                                        ;indicador.


;a rotina janela é a responsável pelo desenho da própria
;janela no vídeo

janela
                push    bc              ;salva todos os registros
                push    de              ;alterados pela rotina
                push    hl              ;
                call    posit           ;posiciona o cursor em x1,y1
                ld      a,d             ;a=x2
                sub     h               ;a=x2-x1
                dec     a               ;a=núm. de pos. da linha do
                                        ;topo
                ld      b,a             ;b=núm. de pos. da linha do
                                        ;topo
                push    bc              ;salva núm. pos. da linha do
                                        ;topo
                ld      a,#18           ;a=carac. do canto sup. esq.
                out     (#98),a         ;escreve o caractere
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
janel1          ld      a,#17           ;a=traço horizontal
                out     (#98),a         ;escreve a linha superior
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
                djnz    janel1          ;
                ld      a,#19           ;a=caractere do canto sup.
                                        ;direito
                out     (#98),a         ;escreve o caractere
                pop     bc              ;recupera núm. pos. linha do
                                        ;topo
                ld      l,e             ;l=y2
                call    posit           ;coloca o cursor em x1,y2
                ld      a,#1a           ;a=caractere do canto inf.
                                        ;esquerdo
                out     (#98),a         ;escreve o caractere
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
janel2          ld      a,#17           ;a=traço horizontal
                out     (#98),a         ;escreve a linha inferior
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
```

```
          djnz    janel2          ;
          ld      a,#1b           ;a=caractere do canto inf.
                                  ;direito
          out     (#98),a         ;escreve o caractere
          pop     hl              ;recupera x1,y1
          pop     de              ;recupera x2,y2
          push    de              ;salva x2,y2
          push    hl              ;salva x1,y1
          ld      a,e             ;a=y2
          sub     l               ;a=y2-y1
          dec     a               ;a=núm. de posições verticais
          ld      b,a             ;b=núm. de posições verticais
          ld      a,d             ;a=x2
          sub     h               ;a=x2-x1
          dec     a               ;a=núm. de pos. horizontais
          ld      c,a             ;c=núm. de pos. horizontais
          inc     l               ;y1=y1+1
janel3    call    posit           ;posiciona o cursor
          ld      a,#16           ;a=traço vertical
          out     (#98),a         ;escreve o caractere
          push    bc              ;salva núm. de pos. verticais
          ld      b,c             ;b=núm. de pos. horizontais
janel4    ld      a,#20           ;a=caractere espaço em branco
          out     (#98),a         ;escreve o carac. para limpar
          ex      (sp),hl         ;demora para sincronização
          ex      (sp),hl         ;
          djnz    janel4          ;a janela
          ld      a,#16           ;a=traço vertical
          out     (#98),a         ;escreve o caractere
          inc     l               ;y1=y1+1
          pop     bc              ;recupera núm. de pos.
                                  ;verticais
          djnz    janel3          ;repete até acabar as linhas
                                  ;verticais
          pop     hl              ;recupera os registros salvos
          pop     de              ;
          pop     bc              ;
          ret                     ;e retorna

tabwindow

          defb    #00             ;coordenada x1
          defb    #00             ;coordenada y1
          defb    #00             ;coordenada x2
          defb    #00             ;coordenada y2
          defb    #00             ;tipo de apresentação
```

```
bufcol          defb    #00             ;buffer auxiliar
buflin          defb    #00             ;buffer auxiliar
coror1          defw    #0000           ;buffer auxiliar
coror2          defw    #0000           ;buffer auxiliar


;a rotina calpadvid calcula os padrões do vídeo: tabela de
; padrões e o número de colunas. Os valores resultantes são
;colocados em tabpad e colunas, respectivamente.

calpadvid
                ex      af,af'          ;salva o registro af
                exx                     ;salva os demais registros
                ld      hl,(txtcgp)     ;obtém a tab. de padrões
                ld      a,(versao)      ;obtém a versão
                or      a               ;do MSX. É MSX1?
                jr      z,calpad_1      ;Sim, vai para calpad_1
                ld      a,(linlen)      ;obtém o tamanho da tela
                cp      41              ;O MSX2 está em 80 colunas?
                jr      c,calpad_1      ;Não, vai para calpad_1
                ld      a,80            ;Sim, a=80 colunas
                add     hl,hl           ;calcula novo end. da
                                        ;tabela de padrões
                jr      calpad_2        ;vai para calpad_2
calpad_1        ld      a,40            ;a=40 colunas
calpad_2        ld      (colunas),a     ;salva as colunas
                ld      (tabpad),hl     ;salva o end. da tab. de
                                        ;padrões
                exx                     ;recupera os registros salvos
                ex      af,af'          ;
                ret                     ;retorna


;a rotina lefrase transporta da tela para buffer bufnor
;a sentença a inverter
lefrase
                push    af              ;salva os registros
                push    bc              ;modificados por esta
                push    de              ;rotina
                push    hl              ;
                call    calendfra       ;calcula o end. da string
                                        ;na VRAM
                call    setvdprd        ;prepara o VDP para leitura
                ld      b,(ix+#00)      ;obtém o comp. da string
                ld      hl,bufnor+#03   ;hl->início da string
```

```
                call    lelinha          ;lê a string
                pop     hl               ;recupera os registros
                pop     de               ;salvos
                pop     bc               ;
                pop     af               ;
                ret                      ;retorna


;a rotina calendfra,baseada nas coordenadas fornecidas, calcula o endereço da sentença a
;inverter na memória VRAM

calendfra
                ld      h,#00            ;calcula o end. da string
                ld      d,h              ;na memória VRAM, baseada
                ld      a,(grpacy)       ;nas coordenadas passadas
                ld      l,a              ;
                or      a                ;a string está na linha 0?
                jr      z,calend1        ;Sim, vai para calend1
                ld      l,h              ;Não, calcula o end. do
                ld      b,a              ;início da linha
                ld      a,(colunas)      ;
                ld      e,a              ;
loopcal_1       add     hl,de            ;
                djnz    loopcal_1        ;
calend1         ld      a,(grpacx)       ;obtém a coord. x
                ld      e,a              ;e=coord. x
                add     hl,de            ;soma ao end. já encontrado
                ld      (bufnor+#01),hl  ;salva no buffer das strings
                ld      (bufinv+#01),hl  ;normal e invertida
                ret                      ;retorna


;a rotina posit posiciona o cursor nas coordenadas passadas
;por hl. h=x e l=y

posit
                push    af               ;salva todos os
                push    bc               ;registros afetados
                push    de               ;por esta rotina
                push    hl               ;
                ex      de,hl            ;troca o conteúdo de hl pelo
                                         ;do registro de
                ld      hl,#0000         ;zera hl
                ld      a,e              ;a=linha
                or      a                ;é igual a zero?
```

```
                jr      z,posit2        ;Sim, vai para posit2
                push    de              ;Não, salva as coordenadas
                ld      b,a             ;b=núm. da linha
                ld      a,(colunas)     ;a=núm. de colunas
                ld      e,a             ;e=núm. de colunas
                ld      d,#00           ;de=núm. de colunas
posit1          add     hl,de           ;hl=hl+núm. de colunas
                djnz    posit1
                pop     de              ;recupera as coordenadas
posit2          ld      a,d             ;a=coluna
                or      a               ;é igual a zero?
                jr      z,posit3        ;Sim, vai para posit3
                ld      e,a             ;Não, e=coluna
                ld      d,#00           ;de=coluna
                add     hl,de           ;hl aponta para o end. da
                                        ;VRAM
                                        ;correspondente a x1,y1
posit3          call    setvdpwt        ;prepara o VDP para escrita
                pop     hl              ;recupera os registros
                pop     de              ;
                pop     bc              ;
                pop     af              ;
                ret                     ;e retorna


;início da lista com os nomes dos novos comandos e
;endereços de chamada

lista
                defm    "REVERSE"       ;nome do comando
                defb    #00             ;fim do nome
                defw    inverte         ;end. da rotina
                defm    "CLRSCR"        ;nome do comando
                defb    #00             ;fim do nome
                defw    newcls          ;end. da rotina
                defm    "WINDOW"        ;nome do comando
                defb    #00             ;fim do nome
                defw    window          ;end. da rotina
endlista        defb    #00             ;fim da lista


;rotinas para o acesso direto à RAM de vídeo

rdvram
                call    setvdprd        ;ajusta o VDP para leitura
```

```
          in      a,(#98)      ;lê um byte
          ret                  ;retorna

wtvram
          push    af           ;salva o dado a enviar
          call    setvdpwt     ;ajusta o VDP para escrita
          pop     af           ;recupera o dado a enviar
          out     (#98),a      ;envia
          ret                  ;retorna

setvdprd
          ld      a,(versao)   ;obtém a versão do MSX
          or      a            ;é MSX1?
          jr      z,rdvram1    ;Sim, vai para rdvram1
          xor     a            ;Não, inicializa o VDP
          out     (#99),a      ;do MSX2
          ld      a,#8e
          out     (#99),a      ;
rdvram1   ld      a,l          ;informa ao
          out     (#99),a      ;VDP o endereço na
          ld      a,h          ;VRAM onde será
          and     #3f          ;lido o dado
          out     (#99),a      ;
          ex      (sp),hl      ;demora para
          ex      (sp),hl      ;sincronização
          ret

setvdpwt
          ld      a,(versao)   ;obtém a versão do MSX
          or      a            ;é MSX1?
          jr      z,wtvram1    ;Sim, vai para wtvram1
          xor     a            ;Não, inicializa o VDP
          out     (#99),a      ;do MSX2
          ld      a,#8e
          out     (#99),a
wtvram1   ld      a,l          ;informa ao
          out     (#99),a      ;VDP o endereço na
          ld      a,h          ;VRAM onde o
          and     #3f          ;dado será
          or      #40          ;gravado
          out     (#99),a      ;
          ex      (sp),hl      ;demora para
          ex      (sp),hl      ;sincronização
          ret                  ;retorna
```

```
;a rotina lelinha transporta uma linha da VRAM para a RAM

lelinha
                in      a,(#98)         ;lê o carac. da VRAM
                ld      (hl),a          ;salva-o no buffer apontado
                                        ;por hl
                inc     hl              ;incrementa o ponteiro do
                                        ;buffer
                djnz    lelinha         ;prepara a próxima leitura na
                                        ;tela
                ret                     ;retorna


;a rotina esclinha transporta uma linha da RAM para a VRAM

esclinha
                ld      a,(hl)          ;lê o carac. do buffer
                out     (#98),a         ;escreve-o na VRAM
                inc     hl              ;incrementa o ponteiro do
                                        ;buffer
                djnz    esclinha        ;prepara a próxima escrita na
                                        ;tela
                ret                     ;retorna


;área das variáveis usadas no programa

bufnor
                defb    #00             ;tamanho da string
                defw    #0000           ;posição de escrita
                defs    33              ;string
bufinv
                defb    #00             ;tamanho da string
                defw    #0000           ;posição de escrita
                defs    33              ;string

inversao        defb    #00             ;flag de campo já invertido

compstr         defb    #00             ;comprimento da string

colunas         defb    #00             ;número de colunas na tela

tabpad          defw    #0000           ;endereço da tab. de padrões

auxiliar        defw    #0000           ;ponteiro do BASIC
```

```
bufferlinha
            defs    81                  ;buffer de linha da tela

fim         equ     $
```

Listagem em linhas DATA do código-objeto do programa que implementa o comando WINDOW:

```
10 FOR A%=&HD000 TO &HD52F
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG24.BIN",&HD000,&HD52F
100 DATA F3,DB,A8,47,E6,F0,0F,0F,0F,0F,B0,D3,A8,E6,03,32
110 DATA 2E,D0,78,21,2D,D0,11,B1,FF,01,05,00,ED,B0,21,32
120 DATA D0,11,00,40,01,FD,04,ED,B0,D3,A8,FB,C9,F7,00,00
130 DATA 40,C9,F5,C5,D5,E5,7B,FE,02,C2,20,40,2A,AF,F6,7E
140 DATA B7,C2,16,40,23,23,23,23,DD,21,66,46,CD,AD,40,CD
150 DATA 25,40,E1,D1,C1,F1,C9,22,A9,44,21,F1,43,01,1C,00
160 DATA ED,5B,A9,44,1A,ED,B1,E2,48,40,13,1A,BE,C2,2E,40
170 DATA 23,7E,FE,00,CA,49,40,C3,38,40,C9,23,4E,23,46,F1
180 DATA C5,ED,53,A9,44,EB,C9,DD,E5,DD,E5,DD,21,66,46,CD
190 DATA AD,40,DD,E1,28,1A,FE,2C,28,12,2B,DD,E5,C5,DD,21
200 DATA 0E,52,CD,AD,40,C1,DD,E1,7B,DD,77,00,DD,23,10,D9
210 DATA DD,E1,C9,3E,05,32,14,F4,C3,89,40,E1,D1,C1,F1,3A
220 DATA 14,F4,5F,C9,2B,AF,32,14,F4,D1,D1,C1,F1,D1,C1,F1
230 DATA DD,E1,DD,E1,F5,C5,D5,DD,21,66,46,C3,AD,40,C9,CD
240 DATA 59,01,C9,06,04,DD,21,83,41,CD,55,40,DD,7E,00,32
250 DATA B7,FC,DD,7E,01,32,B9,FC,DD,7E,02,32,A5,44,FE,21
260 DATA D2,81,40,22,A9,44,DD,7E,03,B7,28,16,3A,A4,44,B7
270 DATA 28,10,2A,5D,44,CD,34,44,21,5F,44,3A,5C,44,47,CD
280 DATA 55,44,3E,FF,32,A4,44,2A,A9,44,3A,A5,44,B7,CA,92
290 DATA 40,DD,21,5C,44,FD,21,80,44,DD,77,00,FD,77,00,CD
300 DATA 70,43,3A,B7,FC,5F,3A,A6,44,BB,DA,81,40,3A,B9,FC
310 DATA FE,18,D2,81,40,CD,92,43,F3,2A,A7,44,E5,11,00,07
320 DATA 19,EB,0E,E0,DD,46,00,DD,21,5F,44,FD,21,83,44,E1
330 DATA C5,E5,D5,11,08,00,DD,46,00,19,10,FD,D1,06,08,CD
340 DATA 0E,44,2F,EB,CD,14,44,EB,23,13,10,F3,E1,C1,FD,71
350 DATA 00,0C,DD,23,FD,23,10,D8,FD,21,80,44,FD,6E,01,FD
360 DATA 66,02,CD,34,44,FD,46,00,21,83,44,CD,55,44,FB,2A
370 DATA A9,44,C3,92,40,00,00,00,00,06,02,DD,21,44,42,CD
380 DATA 55,40,22,A9,44,DD,7E,00,FE,02,D2,81,40,CD,70,43
390 DATA 3A,A6,44,5F,DD,7E,01,1C,BB,D2,81,40,B7,CA,92,40
400 DATA DD,7E,00,B7,C2,FB,41,F3,DD,46,01,C5,2A,B3,F3,3A
410 DATA A6,44,5F,16,00,3A,B1,F3,47,C5,CD,1C,44,3A,A6,44
420 DATA D5,E5,21,AB,44,47,CD,4E,44,3E,20,77,E1,D1,CD,34
```

```
430 DATA 44,D5,E5,21,AC,44,3A,A6,44,47,CD,55,44,E1,D1,19
440 DATA C1,10,D6,C1,10,C5,FB,2A,A9,44,C3,92,40,F3,DD,46
450 DATA 01,C5,2A,B3,F3,3A,A6,44,5F,16,00,3A,B1,F3,47,C5
460 DATA CD,1C,44,3A,A6,44,D5,E5,21,AC,44,47,CD,4E,44,3E
470 DATA 20,21,AB,44,77,E1,D1,CD,34,44,D5,E5,21,AB,44,3A
480 DATA A6,44,47,CD,55,44,E1,D1,19,C1,10,D3,C1,10,C2,FB
490 DATA 2A,A9,44,C3,92,40,00,00,DD,21,65,43,06,05,CD,55
500 DATA 40,22,A9,44,CD,70,43,DD,21,65,43,DD,66,00,DD,6E
510 DATA 01,DD,56,02,DD,5E,03,7C,BA,D2,81,40,7D,BB,D2,81
520 DATA 40,7B,FE,18,D2,81,40,3A,A6,44,3D,BA,DA,81,40,DD
530 DATA 7E,04,B7,C2,8D,42,CD,07,43,2A,A9,44,C3,92,40,AF
540 DATA 32,6A,43,32,6B,43,22,6C,43,ED,53,6E,43,7C,82,CB
550 DATA 3F,3D,67,3C,3C,57,7D,83,CB,3F,3D,6F,3C,3C,5F,CD
560 DATA 07,43,CD,CE,42,CD,DD,42,CD,07,43,CD,EC,42,CD,F9
570 DATA 42,20,EF,2A,6C,43,ED,5B,6E,43,CD,07,43,C3,87,42
580 DATA 3A,6D,43,BC,30,03,25,14,C9,3E,01,32,6A,43,C9,3A
590 DATA 6C,43,BD,30,03,2D,1C,C9,3E,01,32,6B,43,C9,F5,E5
600 DATA 21,00,10,2B,7C,B5,20,FB,E1,F1,C9,3A,6A,43,FE,01
610 DATA 28,01,C9,3A,6B,43,FE,01,C9,C5,D5,E5,CD,C9,43,7A
620 DATA 94,3D,47,C5,3E,18,D3,98,E3,E3,3E,17,D3,98,E3,E3
630 DATA 10,F8,3E,19,D3,98,C1,6B,CD,C9,43,3E,1A,D3,98,E3
640 DATA E3,3E,17,D3,98,E3,E3,10,F8,3E,1B,D3,98,E1,D1,D5
650 DATA E5,7B,95,3D,47,7A,94,3D,4F,2C,CD,C9,43,3E,16,D3
660 DATA 98,C5,41,3E,20,D3,98,E3,E3,10,F8,3E,16,D3,98,2C
670 DATA C1,10,E7,E1,D1,C1,C9,00,00,00,00,00,00,00,00,00
680 DATA 00,00,08,D9,2A,B7,F3,3A,2D,00,B7,28,0C,3A,B0,F3
690 DATA FE,29,38,05,3E,50,29,18,02,3E,28,32,A6,44,22,A7
700 DATA 44,D9,08,C9,F5,C5,D5,E5,CD,AA,43,CD,1C,44,DD,46
710 DATA 00,21,5F,44,CD,4E,44,E1,D1,C1,F1,C9,26,00,54,3A
720 DATA B9,FC,6F,B7,28,09,6C,47,3A,A6,44,5F,19,10,FD,3A
730 DATA B7,FC,5F,19,22,5D,44,22,81,44,C9,F5,C5,D5,E5,EB
740 DATA 21,00,00,7B,B7,28,0C,D5,47,3A,A6,44,5F,16,00,19
750 DATA 10,FD,D1,7A,B7,28,04,5F,16,00,19,CD,34,44,E1,D1
760 DATA C1,F1,C9,52,45,56,45,52,53,45,00,B1,40,43,4C,52
770 DATA 53,43,52,00,87,41,57,49,4E,44,4F,57,00,46,42,00
780 DATA CD,1C,44,DB,98,C9,F5,CD,34,44,F1,D3,98,C9,3A,2D
790 DATA 00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C,E6
800 DATA 3F,D3,99,E3,E3,C9,3A,2D,00,B7,28,07,AF,D3,99,3E
810 DATA 8E,D3,99,7D,D3,99,7C,E6,3F,F6,40,D3,99,E3,E3,C9
820 DATA DB,98,77,23,10,FA,C9,7E,D3,98,23,10,FA,C9,00,00
830 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
840 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
850 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
860 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
870 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
880 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
890 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
```

```
900 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
910 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
920 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,29,09
```

Antes de passar às listagens do programa que implementa o comando **MENU**, gostaria de fazer algumas observações importantes. Assim sendo, vamos começar por um resumo das rotinas da BIOS do MSX empregadas neste novo comando. Como você já sabe, os dois modelos de MSX (1.0 e 2.0) são compatíveis a nível de **BIOS**, o que implica em afirmar que devemos, na medida do possível, dar prioridade ao uso das rotinas contidas na BIOS do MSX. Francamente, a única exceção está nas rotinas para vídeo, onde prefiro usar o acesso direto, tendo em vista a lentidão dessas rotinas na BIOS; fora esta pequena exceção, é muito mais vantajoso usar as rotinas da BIOS pelo simples fato de já estarem prontas e funcionando sem problemas. Desta forma, vamos fazer uso das rotinas listadas na Tabela 3.1, abaixo.

Nome da rotina	**: GTSTCK**
Endereço inicial	**: #00D5**
Modo de chamada	**: CALL**
Parâmetros de entrada	**: A=Identificação do joystick 0=teclas do cursor 1=joystick A 2=joystick B**
Parâmetros de saída	**: A=Código do posicionamento do joystick**
Registros alterados	**: AF,B,DE,HL**
Nome da rotina	**: GTTRIG**
Endereço inicial	**: #00D8**
Modo de chamada	**: CALL**
Parâmetros de entrada	**: A=Identificação do disparo 0=barra de espaço 1=disparo 1 do joystick A 2=disparo 1 do joystick B 3=disparo 2 do joystick A 4=disparo 2 do joystick B**
Parâmetros de saída	**: A=Código do status**
Registros alterados	**: AF,BC**
Nome da rotina	**: KILBUF**
Endereço inicial	**: #0156**
Modo de chamada	**: CALL**
Parâmetros de entrada	**: Nenhum**
Parâmetros de saída	**: Nenhum**
Registros alterados	**: HL**
Nome da rotina	**: VARSRC**

Tabela 3.1: Rotinas utilizadas no comando **MENU**

Endereço inicial	**: #5EA4**
Modo de chamada	**: Por intermédio de CALBAS**
Parâmetros de entrada	**: HL=aponta para o primeiro caractere do nome da variável**
Parâmetros de saída	**: HL=aponta para o caractere após o nome da variável DE=endereço da variável**
Registros alterados	**: AF,BC,DE,HL**

Tabela 3.1: Rotinas utilizadas no comando **MENU** (continuação)

Como já mencionei anteriormente, você deve consultar o "***Livro Vermelho do MSX***" para tirar quaisquer dúvidas sobre o funcionamento das rotinas acima. Em todo caso, as funções desempenhadas por cada uma delas são as seguintes:

GTSTCK Lê o posicionamento dos joysticks ou teclas do cursor.

GTTRIG Lê o estado dos botões de tiro dos joysticks e da barra de espaço.

KILBUF Limpa o buffer do teclado.

VARSRC Procura o endereço inicial de uma variável do BASIC apontada pelo registro HL.

Devido à sua simplicidade, creio que o uso das rotinas acima ficará claro examinando-se os comentários que acompanham a listagem do código-fonte. Vamos então às listagens.

Listagem em assembly Z-80 do código-fonte do programa que implementa o comando MENU:

```
;programa que implementa o comando MENU
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG25.BIN=PROG25.GEN
;
;onde PROG25.GEN é o nome do arquivo-texto com esta
;listagem


versao          equ     #002d
gtstck          equ     #00d5
gttrig          equ     #00d8
kilbuf          equ     #0156
```

```
linlen          equ     #f3b0
txtnam          equ     #f3b3
txtcgp          equ     #f3b7
calbas          equ     #0159
chrgtr          equ     #4666
evlexp          equ     #520e
getcrd          equ     #579c
varsrc          equ     #5ea4
dridef          equ     #f247
crtcnt          equ     #f3b1
errflg          equ     #f414
valtyp          equ     #f663
savtxt          equ     #f6af
scrmod          equ     #fcaf
grpacx          equ     #fcb7
grpacy          equ     #fcb9
hkeyl           equ     #fd9a
herro           equ     #ffb1

                defb    #fe                     ;simula em CP/M
                defw    inicio                  ;o cabecalho de
                defw    fim-erro+rotina+#01
                                                ;um arquivo
                defw    inicio                  ;.BIN

                org     #d000

inicio
                di                              ;desabilita as interrupções
                in      a,(#a8)                 ;lê a atual configuração dos
                ld      b,a                     ;slots e prepara para ativar
                and     #f0                     ;a página 1 do slot da RAM
                rrca                            ;
                rrca                            ;
                rrca                            ;
                rrca                            ;
                or      b                       ;
                out     (#a8),a                 ;ativa as páginas 1, 2 e 3 em
                and     #03                     ;RAM e descobre o slot onde
                ld      (slot),a                ;se encontram os 64Kb de RAM
                ld      a,b                     ;a=config. inicial dos slots
                ld      hl,novohook             ;promove o desvio do hook
                ld      de,herro                ;dos erros
                ld      bc,0005                 ;
                ldir                            ;
                ld      hl,rotina               ;transfere a rotina para a
```

```
                ld      de,erro         ;página 1 da RAM
                ld      bc,fim-erro+#01 ;
                ldir                    ;
                out     (#a8),a         ;habilita a config. original
                ei                      ;habilita as interrupções
                ret                     ;retorna ao BASIC


novohook
                defb    #f7             ;RST #30
slot            defb    #00             ;identificação do slot
                defw    erro            ;endereço de desvio
                defb    #c9             ;RET



rotina
                org     #4000

erro
                push    af              ;salva os registros afetados
                push    bc              ;
                push    de              ;
                push    hl              ;
examerro        ld      a,e             ;a=código do erro
                cp      #02             ;é erro de sintaxe?
                jp      nz,aborta       ;Não, volta ao BASIC
pegachar        ld      hl,(savtxt)     ;Sim, obtém a posição do
                                        ;ponteiro do BASIC
                ld      a,(hl)          ;é final de linha?
                or      a               ;
                jp      nz,proxchar     ;Não, obtém o próximo carac.
pula            inc     hl              ;Sim, pula as 4 posições
                inc     hl              ;referentes ao número da
                inc     hl              ;linha e ao end. da próxima
                inc     hl              ;linha
proxchar        ld      ix,chrgtr       ;obtém o primeiro caractere
                call    chamabasic      ;do novo comando
compara         call    compstring      ;compara com a lista de novos
                                        ;comandos
aborta          pop     hl              ;recupera os registros salvos
                pop     de              ;
                pop     bc              ;
                pop     af              ;
                ret                     ;retorna ao BASIC
```

```
;a rotina compstring é a responsável pela localização da
;rotina do novo comando

compstring
            ld      (auxiliar),hl       ;salva o pont. do BASIC
            ld      hl,lista            ;hl aponta p/ lista de
                                                  ;comandos
            ld      bc,endlista-lista
                                        ;bc=tamanho da lista
compstr1    ld      de,(auxiliar)       ;de aponta p/ comando em
                                        ;BASIC
            ld      a,(de)              ;obtém o primeiro byte
            cpir                        ;tenta encontrá-lo na
                                        ;lista
            jp      po,naoachei         ;se bc=0 -> não achou
compstr2    inc     de                  ;achou o primeiro e
                                        ;parte
            ld      a,(de)              ;para a comparação dos
            cp      (hl)                ;demais caracteres. Vai
                                        ;para
            jp      nz,compstr1         ;compstr1 se achar um
                                        ;byte diferente.
            inc     hl                  ;Caso contrário,
            ld      a,(hl)              ;continua até o fim do
                                        ;comando.
            cp      #00                 ;Se chegou ao fim e,
            jp      z,achei             ;então, achou o comando
                                        ;na lista.
            jp      compstr2            ;Caso contrário, compara
naoachei    ret                         ;o próximo byte.
achei       inc     hl                  ;Se achou, obtém o
                                        ;endereço
            ld      c,(hl)              ;de entrada da rotina
            inc     hl                  ;colocando-o em BC
            ld      b,(hl)              ;
            pop     af                  ;retira da pilha o
                                        ;endereço de
            push    bc                  ;retorno para a rotina
                                        ;erro e
                                        ;coloca na pilha o
                                        ;endereço de
                                        ;entrada da rotina do
                                        ;novo comando
            ld      (auxiliar),de       ;guarda o ponteiro
                                        ;do BASIC
            ex      de,hl               ;transfere o ponteiro
                                        ;para HL
```

```
                ret                          ;usa RET para dar um
                                             ;jump
                                             ;para o endereço contido
                                             ;BC que foi salvo na
                                             ;pilha


;a rotina lecomandos é a responsável pela interpretação dos
;valores que se seguem ao nome do novo comando. Os valores
;devem estar separados por vírgulas. O par IX na entrada
;aponta para uma tabela com valores default e o registro B
;contém o número de valores a interpretar

lecomandos
                push    ix                   ;salva o par IX
lecoman_1       push    ix                   ;
                ld      ix,chrgtr            ;obtém um valor
                call    chamabasic
                pop     ix                   ;recupera o ponteiro IX
                jr      z,lecoman_4          ;Se não houver valor, vai para
                                             ;lecoman_4
                cp      ","                  ;é vírgula?
                jr      z,lecoman_3          ;Sim, assume o default
                dec     hl                   ;Não, obtém o valor
                push    ix                   ;salva o ponteiro
                push    bc                   ;salva o contador
                ld      ix,evlexp            ;obtém o valor
                call    chamabasic           ;
                pop     bc                   ;recupera o valor
                pop     ix                   ;recupera o ponteiro
                ld      a,e                  ;a=valor interpretado
                ld      (ix+#00),a           ;coloca-o na tabela
lecoman_3       inc     ix                   ;incrementa o ponteiro
                djnz    lecoman_1            ;repete para os demais
                                             ;valores
lecoman_4       pop     ix                   ;recupera o ponteiro salvo
                ret                          ;retorna


;a rotina erro_05 emite a mensagem de chamada ilegal

erro_05
                ld      a,#05                ;a=código do erro
                ld      (errflg),a           ;salva o erro em errflg
                jr      voltaerro            ;vai para voltaerro
```

```
;a rotina erro_13 emite a mensagem de tipo incompatível

erro_13
                ld      a,13            ;a=código do erro
                ld      (errflg),a      ;salva o erro em errflg
                jr      voltaerro       ;vai para voltaerro


;a rotina voltaerro substitui o erro original por um outro

voltaerro
                pop     hl              ;recupera os pares
                pop     de              ;salvos no início da
                pop     bc              ;rotina erro
                pop     af              ;
                ld      a,(errflg)      ;a=erro a ser emitido
                ld      e,a             ;e=erro a ser emitido
                ret                     ;volta para a rotina
                                        ;de manipulação de erros
                                        ;do interpretador BASIC


;a rotina volta promove á volta ao BASIC sem que haja
;interrupção no processamento, ou seja, sem que o sistema
;detecte o erro ocorrido

volta
                dec     hl              ;hl aponta para o final
                                        ;do último comando
                xor     a               ;modifica para nenhum erro
                ld      (errflg),a      ;ocorrido
                pop     de              ;recupera os registros
                pop     de              ;salvos no início da rotina
                pop     bc              ;erro. Note que hl não é
                pop     af              ;recuperado.
                pop     de              ;obtém o end. de retorno da
                                        ;nossa rotina para a rotina
                                        ;de chaveamento de slots
                pop     bc              ;obtém o end. de retorno da
                                        ;rotina de chaveamento de
                                        ;slots para RST #30
                pop     af              ;obtém o end. de retorno de
                                        ;RST #30 para o hook
                pop     ix              ;obtém/destrói o end. de
                                        ;retorno do hook para a
                                        ;rotina
                                        ;de erro do interpretador
```

```
                                        ;BASIC
              pop     ix                ;obtém/destrói o end. de
                                        ;retorno
                                        ;da rotina de erro para o
                                        ;interpretador BASIC
              push    af                ;repõe na pilha os end. de
              push    bc                ;retorno salvos nestes
              push    de                ;registros
              ld      ix,chrgtr         ;faz com que hl aponte
              jp      chamabasic        ;para o próximo comando
              ret                       ;e retorna ao BASIC

chamabasic
              call    calbas            ;chama a rotina calbas
              ret                       ;retorna

inverte
              ld      b,#04             ;b=núm. de valores
              ld      ix,tabinver       ;ix=tabela default
              call    lecomandos        ;lê os 4 valores
              ld      (auxiliar),hl     ;salva ptr. do BASIC
              call    rotinversao       ;chama a rotina de
                                        ;inversao
              ld      hl,(auxiliar)     ;recupera ptr. do BASIC
              jp      volta             ;volta para o BASIC

rotinversao
              ld      a,(ix+#00)        ;obtém coord. x
              ld      (grpacx),a        ;salva em grpacx
              ld      a,(ix+#01)        ;obtém coord. y
              ld      (grpacy),a        ;salva em grpacy
              ld      a,(ix+#02)        ;obtém o comp. da string
              ld      (compstr),a       ;salva em compstr
              cp      33                ;é >= 33?
              jp      nc,erroinv_05     ;Sim, chamada ilegal
              ld      a,(ix+#03)        ;a=flag de recuperação
              or      a                 ;está setada?
              jr      z,inverte0        ;Não, vai para inverte0
              ld      a,(inversao)      ;Sim, já foi feita alguma
              or      a                 ;inversão?
              jr      z,inverte0        ;Não, vai para inverte0
              ld      a,(bufnor)        ;Sim, obtém o comprimento
                                        ;da string.
              or      a                 ;é zero?
              jr      z,inverte0        ;Sim, vai para inverte0
```

```
          ld      hl,(bufnor+#01)  ;Não, obtém a pos. da string
          call    setvdpwt         ;antiga e prepara o VDP
          ld      hl,bufnor+#03    ;hl->início da string antiga
          ld      a,(bufnor)       ;a=comp. da string antiga
          ld      b,a              ;a=comp. da string antiga
          call    esclinha         ;escreve a string antiga
                                   ;no modo normal
inverte0
          ld      hl,(auxiliar)    ;hl=ponteiro do BASIC
          ld      a,(compstr)      ;a=comp. da string
          or      a                ;comprimento=0?
          jp      z,voltainv       ;Sim, volta sem erro
          ld      ix,bufnor        ;Não, ajusta os buffers
          ld      iy,bufinv        ;para a string normal e
          ld      (ix+#00),a       ;para a string invertida
          ld      (iy+#00),a       ;com o comprimento da
          call    calpadvid        ;calcula os padrões do vídeo
          ld      a,(grpacx)       ;obtém a coord. x
          ld      e,a              ;coloca em e
          ld      a,(colunas)      ;obtém o número de colunas
          cp      e                ;compara com o valor lido
          jp      c,erroinv_05     ;Se valor lido > colunas
                                   ;volta com erro
          ld      a,(grpacy)       ;obtém a coord. y
          cp      24               ;é maior do que 23?
          jp      nc,erroinv_05    ;Sim, volta com erro
          call    lefrase          ;lê a string a inverter
inverte3  di                       ;desabilita as interrupções
          ld      hl,(tabpad)      ;hl=end. da tab. de padrões
          push    hl               ;salva o end. da tab. de
                                   ;padrões
          ld      de,#e0*8         ;calcula o endereço do
          add     hl,de            ;desenho do caractere #e0
          ex      de,hl            ;de=end. do des. do carac. #e0
          ld      c,#e0            ;c=#e0
          ld      b,(ix+#00)       ;b=número de carac. da string
          ld      ix,bufnor+#03    ;ix=end. da string normal
          ld      iy,bufinv+#03    ;iy=end. da string invertida
          pop     hl               ;recupera hl
loopinv1  push    bc               ;salva bc
          push    hl               ;salva hl
          push    de               ;salva de
          ld      de,#0008         ;bytes para o desenho de cada
                                   ;caractere
          ld      b,(ix+#00)       ;b=caractere a inverter
loopinv2  add     hl,de            ;calcula o endereço desse
          djnz    loopinv2         ;caractere
```

```
              pop     de                ;recupera de
              ld      b,#08             ;prepara a modificação
loopinv3      call    rdvram            ;lê o byte original
              cpl                       ;inverte
              ex      de,hl             ;transfere para o novo
              call    wtvram            ;destino
              ex      de,hl             ;
              inc     hl                ;hl=hl+1
              inc     de                ;de=de+1
              djnz    loopinv3          ;b=b-1
              pop     hl                ;recupera hl
              pop     bc                ;recupera bc
              ld      (iy+#00),c        ;modifica a string
              inc     c                 ;c=c+1
              inc     ix                ;aponta para o próximo carac.
              inc     iy
              djnz    loopinv1          ;repete até o fim da string
envinvert
              ld      iy,bufinv         ;recupera o ponteiro do
                                        ;buffer
              ld      l,(iy+#01)        ;hl=end. de escrita da string
              ld      h,(iy+#02)        ;
              call    setvdpwt          ;ajusta o VDP para escrita
              ld      b,(iy+#00)        ;b=núm. de carac. da string
              ld      hl,bufinv+#03     ;end. da string
              call    esclinha          ;escreve a frase invertida
              ld      a,#ff             ;sinaliza uma inversão
              ld      (inversao),a      ;
              ei                        ;habilita as interrupções
              ret                       ;retorna

erroinv_05
              pop     af                ;destrói o end. de retorno
              jp      erro_05           ;vai para erro_04

voltainv
              pop     af                ;destrói a pilha
              jp      volta             ;volta para o BASIC


tabinver
              defb    #00               ;posição x
              defb    #00               ;posição y
              defb    #00               ;comprimento
              defb    #00               ;flag de restauração
```

```
newcls
              ld      b,#02           ;b=núm. de valores
              ld      ix,tabnewcls    ;ix->tab. de valores default
              call    lecomandos      ;
              ld      (auxiliar),hl   ;salva o ponteiro do BASIC
              ld      a,(ix+#00)      ;obtém o primeiro valor
              cp      #02             ;verifica o limite
              jp      nc,erro_05      ;se estiver fora, volta com
                                       erro
              call    calpadvid       ;calcula os padrões do vídeo
              ld      a,(colunas)     ;obtém o número de colunas
              ld      e,a             ;e=número de colunas
              ld      a,(ix+#01)      ;a=número de rotações
              inc     e
              cp      e               ;rotações>colunas?
              jp      nc,erro_05      ;Sim, volta com erro
              or      a               ;Zero rotações?
              jp      z,volta         ;Sim, volta sem erro
              ld      a,(ix+#00)      ;a=tipo de rotação
              or      a               ;é zero (esquerda)?
              jp      nz,newcls_0     ;Não, vai para newcls_0


;newcls_1 faz o cls com rotação para a esquerda

newcls_1

              di                      ;desabilita as interrupções
              ld      b,(ix+#01)      ;b=número de rotações
loopcls_11
              push    bc              ;salva contador externo
              ld      hl,(txtnam)     ;hl=end. tabela de nomes
              ld      a,(colunas)     ;a=núm. de colunas
              ld      e,a             ;de=núm. de colunas
              ld      d,#00           ;
              ld      a,(crtcnt)      ;a=número de linhas
              ld      b,a             ;b=número de linhas
loopcls_12
              push    bc              ;salva contador intermediário
              call    setvdprd        ;prepara o VDP para leitura
              ld      a,(colunas)     ;a=núm. de colunas
              push    de              ;salva de
              push    hl              ;salva hl
              ld      hl,bufferlinha  ;hl aponta para o buffer
              ld      b,a             ;b=núm. de colunas
              call    lelinha         ;lê uma linha
              ld      a,#20           ;a=carac. espaço em branco
```

```
                ld      (hl),a              ;coloca o espaço na últ. pos.
                pop     hl                  ;recupera hl
                pop     de                  ;recupera de
                call    setvdpwt            ;prepara o VDP para escrita
                push    de                  ;salva de
                push    hl                  ;salva hl
                ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
                ld      a,(colunas)         ;a=núm. de colunas
                ld      b,a                 ;b=núm. de colunas
                call    esclinha            ;envia a linha já rotada
                pop     hl                  ;recupera hl
                pop     de                  ;recupera de
                add     hl,de               ;hl aponta para a próx. linha
                pop     bc                  ;recupera bc
                djnz    loopcls_12          ;repete para as demais linhas
                pop     bc                  ;recupera bc
                djnz    loopcls_11          ;repete até terminar todas
                                            ;as rotações
                ei                          ;habilita as interrupções
                ld      hl,(auxiliar)       ;recupera o ponteiro do BASIC
                jp      volta               ;retorna ao BASIC


;newcls_0 faz o cls com rotação para a direita

newcls_0
                di                          ;desabilita as interrupções
                ld      b,(ix+#01)          ;b=número de rotações
loopcls_01
                push    bc                  ;salva contador externo
                ld      hl,(txtnam)         ;hl=end. tabela de nomes
                ld      a,(colunas)         ;a=núm. de colunas
                ld      e,a                 ;de=núm. de colunas
                ld      d,#00               ;
                ld      a,(crtcnt)          ;a=número de linhas
                ld      b,a                 ;b=número de linhas
loopcls_02
                push    bc                  ;salva contador intermediário
                call    setvdprd            ;prepara o VDP para leitura
                ld      a,(colunas)         ;a=núm. de colunas
                push    de                  ;salva de
                push    hl                  ;salva hl
                ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
                ld      b,a                 ;b=núm. de colunas
                call    lelinha             ;lê uma linha
                ld      a,#20               ;a=carac. espaço em branco
                ld      hl,bufferlinha      ;hl aponta para o buffer
```

```
                ld      (hl),a          ;coloca o espaço na prim.
                                        ;posição
                pop     hl              ;recupera hl
                pop     de              ;recupera de
                call    setvdpwt        ;prepara o VDP para escrita
                push    de              ;salva de
                push    hl              ;salva hl
                ld      hl,bufferlinha  ;hl aponta para o buffer
                ld      a,(colunas)     ;a=núm. de colunas
                ld      b,a             ;b=núm. de colunas
                call    esclinha        ;envia a linha já rotada
                pop     hl              ;recupera hl
                pop     de              ;recupera de
                add     hl,de           ;hl aponta para a próx. linha
                pop     bc              ;recupera bc
                djnz    loopcls_02      ;repete para as demais linhas
                pop     bc              ;recupera bc
                djnz    loopcls_01      ;repete até terminar todas
                                        ;as rotações
                ei                      ;habilita as interrupções
                ld      hl,(auxiliar)   ;recupera o ponteiro do BASIC
                jp      volta           ;retorna ao BASIC


tabnewcls
                defb    #00             ;tipo de cls
                defb    #00             ;número de rotações


window
                ld      ix,tabwindow    ;ix aponta p/ tabela
                ld      b,#05           ;b=núm. de comandos
                call    lecomandos      ;lê as coordenadas
                ld      (auxiliar),hl   ;salva o ptr. do BASIC
                call    calpadvid       ;calcula os parâmetros
                                        ;do vídeo
                ld      ix,tabwindow    ;ix aponta p/ tabela
                ld      h,(ix+#00)      ;h=x1
                ld      l,(ix+#01)      ;l=y1
                ld      d,(ix+#02)      ;d=x2
                ld      e,(ix+#03)      ;e=y2
                ld      a,h             ;testa se x2>x1
                cp      d               ;
                jp      nc,erro_05      ;x2<=x1->erro
                ld      a,l             ;testa se y2>y1
                cp      e               ;
                jp      nc,erro_05      ;y2<=y1->erro
```

```
                ld      a,e             ;a=y2
                cp      24              ;y2>=24?
                jp      nc,erro_05      ;Sim, volta com erro
                ld      a,(colunas)     ;a=colunas na tela
                dec     a
                cp      d               ;x2>=colunas?
                jp      c,erro_05       ;Sim, volta com erro
                ld      a,(ix+#04)      ;a=tipo de apresentação
                or      a               ;é zero?
                jp      nz,zoom         ;Não, vai para zoom
                call    janela          ;Sim, chama janela

windvolta
                ld      hl,(auxiliar)   ;hl=ponteiro do BASIC
                jp      volta           ;volta para o BASIC


;a rotina zoom cria o efeito de explosão da janela

zoom
                xor     a               ;zera o acumulador
                ld      (bufcol),a      ;zera o buffer da coluna
                ld      (buflin),a      ;zera o buffer da linha
                ld      (coror1),hl     ;salva as coordenadas x1,y1
                ld      (coror2),de     ;salva as coordenadas x2,y2
                ld      a,h             ;carrega a com h (x1)
                add     a,d             ;soma com d (x2) e divide
                srl     a               ;por dois para achar Xmedio
                dec     a               ;decrementa Xmedio
                ld      h,a             ;carrega h (x1) com Xmedio
                inc     a               ;incrementa o ponto médio
                                        ;para que
                inc     a               ;x2=x1+1, onde x1=Xmedio-1
                ld      d,a             ;;carrega d com x2
                ld      a,l             ;a=y1
                add     a,e             ;soma com e (y2) e divide
                srl     a               ;por dois para achar Ymedio
                dec     a               ;decrementa Ymedio
                ld      l,a             ;carrega l (y1) com Ymedio
                inc     a               ;incrementa o ponto médio
                                        ;para que
                inc     a               ;y2=y1+1, onde y1=Ymedio-1
                ld      e,a             ;carrega e com y2
                call    janela          ;desenha a primeira janela
looptest        call    coluna          ;verifica em rel. à coluna-
                                        ;limite
                call    linha           ;verifica em rel. à linha-
```

```
                                        ;limite
            call    janela              ;desenha a janela
            call    espjanela           ;retardo para tornar o efeito
                                        ;visível
            call    teste2              ;verifica se já chegou à
                                        ;janela final
            jr      nz,looptest         ;Não, continua com a explosão
            ld      hl,(coror1)         ;imprime a janela final
            ld      de,(coror2)         ;
            call    janela              ;
            jp      windvolta           ;retorna ao BASIC

coluna      ld      a,(coror1+#01)      ;a=coord. x1 original
            cp      h                   ;já foi atingida?
            jr      nc,fimcol           ;Sim, vai para fimcol
            dec     h                   ;Não, x1=x1-1
            inc     d                   ;x2=x2+1
            ret                         ;retorna
fimcol      ld      a,#01               ;indica que a coluna-limite
            ld      (bufcol),a          ;foi atingida
            ret                         ;retorna

linha       ld      a,(coror1)          ;a=coord. y1 original
            cp      l                   ;já foi atingida?
            jr      nc,fimlin           ;Sim, vai para fimlin
            dec     l                   ;Não, y1=y1-1
            inc     e                   ;y2=y2+1
            ret                         ;retorna
fimlin      ld      a,#01               ;indica que a linha-limite
            ld      (buflin),a          ;foi atingida
            ret                         ;retorna

espjanela   push    af                  ;salva os registros
            push    hl                  ;afetados
            ld      hl,#1000            ;altere este valor para mudar
espjanela1
            dec     hl                  ;o retardo. Decrementa hl
            ld      a,h                 ;verifica se chegou
            or      l                   ;ao fim
            jr      nz,espjanela1       ;Não, volta para espjanela1
            pop     hl                  ;recupera os registros
            pop     af                  ;
            ret                         ;e retorna

teste2      ld      a,(bufcol)          ;verifica se a coluna-
            cp      #01                 ;limite foi atingida
```

```
                jr      z,contes        ;Sim, vai para contes
                ret                     ;Não, retorna
contes          ld      a,(buflin)      ;verifica se a linha-
                cp      #01             ;limite foi atingida
                ret                     ;retorna. Flag Z será o
                                        ;indicador.


;a rotina janela é a responsável pelo desenho da própria
;janela no vídeo

janela
                push    bc              ;salva todos os registros
                push    de              ;alterados pela rotina
                push    hl              ;
                call    posit           ;posiciona o cursor em x1,y1
                ld      a,d             ;a=x2
                sub     h               ;a=x2-x1
                dec     a               ;a=núm. de pos. da linha do
                                        ;topo
                ld      b,a             ;b=núm. de pos. da linha do
                                        ;topo
                push    bc              ;salva núm. pos. da linha do
                                        ;topo
                ld      a,#18           ;a=carac. do canto sup. esq.
                out     (#98),a         ;escreve o caractere
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
janel1          ld      a,#17           ;a=traço horizontal
                out     (#98),a         ;escreve a linha superior
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
                djnz    janel1          ;
                ld      a,#19           ;a=carac. do canto sup. dir.
                out     (#98),a         ;escreve o caractere
                pop     bc              ;recupera núm. pos. linha do
                                        ;topo
                ld      l,e             ;l=y2
                call    posit           ;coloca o cursor em x1,y2
                ld      a,#1a           ;a=carac.do canto inf.  esquerdo
                out     (#98),a         ;escreve o caractere
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
janel2          ld      a,#17           ;a=traço horizontal
                out     (#98),a         ;escreve a linha inferior
                ex      (sp),hl         ;demora para sincronização
```

```
                ex      (sp),hl         ;
                djnz    janel2          ;
                ld      a,#1b           ;a=carac. do canto inf. dir.
                out     (#98),a         ;escreve o caractere
                pop     hl              ;recupera x1,y1
                pop     de              ;recupera x2,y2
                push    de              ;salva x2,y2
                push    hl              ;salva x1,y1
                ld      a,e             ;a=y2
                sub     l               ;a=y2-y1
                dec     a               ;a=núm. de pos. verticais
                ld      b,a             ;b=núm. de pos. verticais
                ld      a,d             ;a=x2
                sub     h               ;a=x2-x1
                dec     a               ;a=núm. de pos. horizontais
                ld      c,a             ;c=núm. de pos. horizontais
                inc     l               ;y1=y1+1
janel3          call    posit           ;posiciona o cursor
                ld      a,#16           ;a=traço vertical
                out     (#98),a         ;escreve o caractere
                push    bc              ;salva núm. de pos. verticais
                ld      b,c             ;b=núm. de pos. horizontais
janel4          ld      a,#20           ;a=caractere espaço em branco
                out     (#98),a         ;escreve o carac. para limpar
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
                djnz    janel4          ;a janela
                ld      a,#16           ;a=traço vertical
                out     (#98),a         ;escreve o caractere
                inc     l               ;y1=y1+1
                pop     bc              ;recupera núm. de pos.
                                        ;verticais
                djnz    janel3          ;repete até acabar as linhas
                                        ;vert.
                pop     hl              ;recupera os registros salvos
                pop     de              ;
                pop     bc              ;
                ret                     ;e retorna


tabwindow
                defb    #00             ;coordenada x1
                defb    #00             ;coordenada y1
                defb    #00             ;coordenada x2
                defb    #00             ;coordenada y2
                defb    #00             ;tipo de apresentação
bufcol          defb    #00             ;buffer auxiliar
```

```
buflin          defb    #00             ;buffer auxiliar
coror1          defw    #0000           ;buffer auxiliar
coror2          defw    #0000           ;buffer auxiliar


;a rotina menu faz a seleção em menus

menu
                ld      ix,tabmenu      ;ix aponta p/ tabela
                ld      b,#06           ;b=núm. de argumentos
                call    lecomandos      ;lê os argumentos
                dec     hl              ;decrementa o ptr. do BASIC
                ld      ix,chrgtr       ;obtém o próximo caractere
                call    chamabasic      ;
                cp      ","             ;é vírgula?
                jp      nz,aborta       ;Não, erro de sintaxe
                ld      ix,chrgtr       ;Sim, obtém o próximo
                call    chamabasic      ;caractere
                jp      z,aborta        ;Se fim de linha->aborta
                ld      ix,varsrc       ;procura a variável
                call    chamabasic      ;inteira
                ld      (endvar),de     ;salva o end. da var.
                ld      a,(valtyp)      ;obtém o tipo da var.
                cp      #02             ;é inteiro?
                jp      nz,erro_13      ;Não, erro de tipo
                ld      (auxiliar),hl   ;salva o ptr. do BASIC
                ld      ix,tabinver     ;ix aponta p/ tabinver
                ld      iy,tabmenu      ;iy aponta p/ tabmenu
                ld      a,(iy+#02)      ;a=x2
                cp      (iy+#00)        ;x2>=x1?
                jp      c,erro_05       ;Não, emite erro
                ld      a,(iy+#03)      ;a=y2
                cp      (iy+#01)        ;y2>=y1?
                jp      c,erro_05       ;Não, emite erro
                call    calpadvid       ;calcula pars. do vídeo
                ld      a,(colunas)     ;a=x1
                dec     a
                cp      (iy+#00)        ;x1>colunas?
                jp      c,erro_05       ;Sim, emite erro
                cp      (iy+#02)        ;x2>colunas?
                jp      c,erro_05       ;Sim, emite erro
                ld      a,24            ;a=núm. máx. de linhas
                cp      (iy+#01)        ;y1>24?
                jp      c,erro_05       ;Sim, emite erro
                cp      (iy+#03)        ;y2>24?
                jp      c,erro_05       ;Sim, emite erro
                ld      h,(iy+#00)      ;h=x1
```

```
                ld      l,(iy+#01)      ;l=y1
                ld      d,(iy+#02)      ;d=x2
                ld      e,(iy+#03)      ;e=y2
                ld      a,(iy+#04)      ;a=comprimento das opções
                ld      (ix+#02),a      ;salva em tabinver
                ld      a,#01           ;a=#01
                ld      (ix+#03),a      ;ajusta a flag de
                                        ;restauração
                ld      (cooratual),hl  ;salva a pos. atual
                xor     a               ;zera a opção atual
                ld      (opcao),a       ;
                ld      (inversao),a    ;zera a inversão
                ld      a,h             ;compara x1 com
                cp      d               ;x2
                jp      z,vertical      ;se x1=x2 mov. vertical
                ld      a,l             ;compara y1 com
                cp      e               ;y2
                jp      nz,aborta       ;se não forem iguais
                                        ;emite erro de sintaxe


;a rotina horizontal controla o mov. horizontal. Na rotina
;horizontal, temos o cálculo do valor da opção final e, a
;partir de horizont_2, temos o controle do movimento
;propriamente dito

horizontal
                ld      a,(compopcao)   ;a=comp. das opções
                ld      e,a             ;salva a em e
                ld      a,(passo)       ;a=incremento entre as
                                        ;opções
                add     a,e             ;a=passo+comprimento
                ld      e,a             ;salva a em e
                ld      a,(coordx1)     ;a=x1
                ld      c,a             ;c=x1
                ld      a,(coordx2)     ;a=x2
                ld      b,#00           ;zera o registro b
horizont_1
                sub     e               ;subtrai a de e
                jr      c,horiz_11      ;Se estourou, sai do loop
                inc     b               ;incrementa b
                cp      c               ;a=x1?
                jr      nz,horizont_1   ;Não, volta para horizont_1
horiz_11        ld      a,b             ;a=opção máxima
                ld      (opcaofim),a    ;salva em opcaofim

horizont_2
```

```
                ld      ix,tabinver       ;ix ponta p/ tabinver
                ld      hl,(cooratual)    ;hl=posição atual
                ld      a,h               ;
                ld      (ix+#00),a        ;
                ld      a,l               ;
                ld      (ix+#01),a        ;
                call    rotinversao       ;faz a inversão
                call    rdstick           ;lê joystick/teclado
                cp      #03               ;o mov. foi para a direita?
                jp      z,movdir          ;Sim, vai para movdir
                cp      #07               ;o mov. foi para a esquerda?
                jp      z,movesq          ;Sim, vai para movesq
                cp      #ff               ;Return ou tiro?
                jp      z,fimsel          ;Sim, vai para fimsel
                jr      horizont_2        ;fecha o loop


;a rotina movdir controla o movimento para a direita

movdir
                ld      a,(passo)         ;a=incremento
                ld      e,a               ;e=a
                ld      a,(xatual)        ;a=posição x atual
                add     a,e               ;soma x atual com o
                                          ;incremento
                ld      e,a               ;e=x atual+passo
                ld      a,(compopcao)     ;a=comp. das opções
                add     a,e               ;a=x atual+passo+comp.
                ld      e,a               ;e=x calculado
                ld      a,(coordx2)       ;a=coord. x limite
                cp      e                 ;x calculado > x limite?
                jr      c,movdir_1        ;Sim, vai para movdir_1
                ld      a,e               ;Não, a=x calculado
                ld      (xatual),a        ;modifica x atual
                ld      a,(opcao)         ;incrementa o valor da
                inc     a                 ;opção
                ld      (opcao),a         ;
                jp      horizont_2        ;volta para horizont_2
movdir_1
                ld      a,(coordx1)       ;hl=coord. original
                ld      h,a               ;
                ld      a,(coordy1)       ;
                ld      l,a               ;
                ld      (cooratual),hl    ;salva em cooratual
                xor     a                 ;zera a opção
                ld      (opcao),a         ;salva em opção
                jp      horizont_2        ;volta para horizont_2
```

```
;a rotina movesq controla o movimento para a esquerda

movesq
            ld      a,(passo)       ;a=incremento
            ld      e,a             ;e=a
            ld      a,(xatual)      ;a=x atual
            ld      c,a             ;salva a em e
            ld      a,(coordx1)     ;a=x1
            cp      c               ;x atual=x1
            jr      z,movesq_1      ;Sim, vai para movesq_1
            ld      a,c             ;Não, calcula nova pos.
            sub     e               ;a=x atual-passo
            ld      e,a             ;e=x atual-passo
            ld      a,(compopcao)   ;a=comp. das opções
            ld      c,a             ;salva a em c
            ld      a,e             ;a=x atual-passo
            sub     c               ;a=x atual-passo-comp
            ld      (xatual),a      ;modifica xatual
            ld      a,(opcao)       ;decrementa o valor
            dec     a               ;da opção
            ld      (opcao),a       ;
            jp      horizont_2      ;volta para horizont_2
movesq_1
            ld      a,(coordx2)     ;hl=coord. original final
            ld      h,a             ;
            ld      a,(coordy2)     ;
            ld      l,a             ;
            ld      (cooratual),hl  ;salva em cooratual
            ld      a,(opcaofim)    ;a=opção final
            ld      (opcao),a       ;salva em opção
            jp      horizont_2      ;fecha o loop


;a rotina vertical controla o movimento vertical. Na rotina
;vertical, temos o cálculo do valor da opção final e, a
;partir de vertical_2, temos o controle do movimento
;propriamente dito

vertical
            ld      a,(passo)       ;a=incremento entre as
                                    ;opções
            ld      e,a             ;salva a em e
            ld      a,(coordy1)     ;a=y1
            ld      c,a             ;c=y1
            ld      a,(coordy2)     ;a=y2
            ld      b,#00           ;zera o registro b
```

```
vertical_1
            sub     e               ;a=a-passo
            inc     b               ;incrementa o registro b
            cp      c               ;a=y1?
            jr      nz,vertical_1   ;Não, volta para vertical_1
            ld      a,b             ;a=opção máxima
            ld      (opcaofim),a    ;salva em opcaofim

vertical_2
            ld      ix,tabinver     ;ix aponta p/ tabinver
            ld      hl,(cooratual)  ;hl=posição atual
            ld      a,h             ;salva a pos. a inverter
            ld      (ix+#00),a      ;
            ld      a,l             ;
            ld      (ix+#01),a      ;
            call    rotinversao     ;faz a inversão
            call    rdstick         ;le joystick/teclado
            cp      #05             ;o movimento foi para baixo?
            jp      z,movbai        ;Sim, vai para movbai
            cp      #01             ;o movimento foi para cima?
            jp      z,movcim        ;Sim, vai para movcim
            cp      #ff             ;Return ou tiro?
            jp      z,fimsel        ;Sim, vai para fimsel
            jp      vertical_2      ;Não, fecha o loop


;a rotina movbai controla o movimento para baixo.

movbai
            ld      a,(passo)       ;a=incremento entre as
                                    ;opções
            ld      e,a             ;salva a em e
            ld      a,(yatual)      ;a=y atual
            add     a,e             ;a=y atual+passo
            ld      e,a             ;salva o resultado em e
            ld      a,(coordy2)     ;a=y2
            cp      e               ;y atual>y2?
            jp      c,movbai_1      ;Sim, vai para movbai_1
            ld      a,e             ;Não, a=y atual
            ld      (yatual),a      ;salva em yatual
            ld      a,(opcao)       ;incrementa a opção
            inc     a               ;atual
            ld      (opcao),a       ;salva em opção
            jp      vertical_2      ;volta para vertical_2

movbai_1
            ld      a,(coordx1)     ;hl=posição inicial
```

```
                ld      h,a                 ;
                ld      a,(coordy1)         ;
                ld      l,a                 ;
                ld      (cooratual),hl      ;salva em cooratual
                xor     a                   ;zera a opção atual
                ld      (opcao),a           ;
                jp      vertical_2          ;volta para vertical_2


;a rotina movcim controla o movimento para cima

movcim
                ld      a,(passo)           ;a=incremento entre as
                                            ;opções
                ld      e,a                 ;salva a em e
                ld      a,(coordy1)         ;a=y1
                ld      c,a                 ;salva y1 em c
                ld      a,(yatual)          ;a=y atual
                cp      c                   ;y atual=y1?
                jr      z,movcim_1          ;Sim, vai para movcim_1
                sub     e                   ;Não, a=y atual-passo
                ld      (yatual),a          ;salva em yatual
                ld      a,(opcao)           ;decrementa o valor da
                dec     a                   ;opção atual
                ld      (opcao),a           ;salva em opcao
                jp      vertical_2          ;volta para vertical_2

movcim_1
                ld      a,(coordx2)         ;hl=posição final
                ld      h,a                 ;
                ld      a,(coordy2)         ;
                ld      l,a                 ;
                ld      (cooratual),hl      ;salva em cooratual
                ld      a,(opcaofim)        ;a=valor da opção final
                ld      (opcao),a           ;salva em opcao
                jp      vertical_2          ;volta para vertical_2


;a rotina fimsel promove a atualização da variável inteira
;e retorna para o BASIC

fimsel
                ld      hl,(endvar)         ;hl=end. da var. inteira
                ld      a,(opcao)           ;a=valor da opção
                ld      (hl),a              ;atualiza a variável
                inc     hl                  ;
                ld      (hl),#00            ;
```

```
                call    kilbuf          ;limpa o buffer do teclado
                ld      hl,(auxiliar)   ;obtém o ptr. do BASIC
                jp      volta           ;volta para o BASIC


tabmenu
coord1
coordx1         defb    #00
coordy1         defb    #00

coord2
coordx2         defb    #00
coordy2         defb    #00

compopcao       defb    #00

passo           defb    #00


;a rotina rdstick lê o estado dos joysticks e das teclas
;do cursor

rdstick
                push    ix              ;salva os registros
                push    iy              ;afetados
                push    bc              ;
                push    de              ;
                push    hl              ;
rdstick_1
                ld      a,#00           ;lê o estado
                call    gtstck          ;das teclas do cursor
                or      a               ;alguma tecla pressionada?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#01           ;lê o estado do
                call    gtstck          ;joystick na porta A
                or      a               ;algum movimento?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#02           ;lê o estado do
                call    gtstck          ;joystick na porta B
                or      a               ;algum movimento?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#00           ;lê o estado da barra
                call    gttrig          ;de espaço
                or      a               ;foi pressionada?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#01           ;lê o estado do botão de
                call    gttrig          ;tiro do joystick A
```

```
                or      a               ;foi pressionado?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#02           ;lê o estado do botão de
                call    gttrig          ;tiro do joystick B
                or      a               ;foi pressionado?
                jr      z,rdstick_1     ;Não, volta para rdstick_1

fimstick        push    af              ;salva o valor lido
                ld      hl,#8000        ;provoca um pequeno
loopendst       dec     hl              ;retardo para que as
                ld      a,h             ;seleções não sejam
                or      l               ;rápidas demais
                jr      nz,loopendst    ;
                pop     af              ;recupera o valor lido
                pop     hl              ;recupera todos os
                pop     de              ;registros salvos
                pop     bc              ;no início da rotina
                pop     iy              ;
                pop     ix              ;
                ret                     ;retorna


;a rotina calpadvid calcula os padrões do vídeo: tabela de
;padrões e o número de colunas. Os valores resultantes são
;colocados em tabpad e colunas, respectivamente.

calpadvid
                ex      af,af'          ;salva o registro af
                exx                     ;salva os demais registros
                ld      hl,(txtcgp)     ;obtém a tab. de padrões
                ld      a,(versao)      ;obtém a versão
                or      a               ;do MSX. É MSX1?
                jr      z,calpad_1      ;Sim, vai para calpad_1
                ld      a,(linlen)      ;obtém o tamanho da tela
                cp      41              ;O MSX2 está em 80 colunas?
                jr      c,calpad_1      ;Não, vai para calpad_1
                ld      a,80            ;Sim, a=80 colunas
                add     hl,hl           ;calcula novo end. da
                                        ;tabela de padrões
                jr      calpad_2        ;vai para calpad_2
calpad_1        ld      a,40            ;a=40 colunas
calpad_2        ld      (colunas),a     ;salva as colunas
                ld      (tabpad),hl     ;salva o end. da tab. de
                                        ;padrões
                exx                     ;recupera os registros salvos
                ex      af,af'          ;
                ret                     ;retorna
```

```
;a rotina lefrase transporta da tela para o buffer bufnor
;a sentença a inverter
lefrase
                push    af              ;salva os registros
                push    bc              ;modificados por esta
                push    de              ;rotina
                push    hl              ;
                call    calendfra       ;calcula o end. da string
                                        ;na VRAM
                call    setvdprd        ;prepara o VDP para leitura
                ld      b,(ix+#00)      ;obtém o comp. da string
                ld      hl,bufnor+#03   ;hl->início da string
                call    lelinha         ;lê a string
                pop     hl              ;recupera os registros
                pop     de              ;salvos
                pop     bc              ;
                pop     af              ;
                ret                     ;retorna


;a rotina calendfra, baseada nas coordenadas fornecidas, calcula o endereço da sentença a
;inverter na memória VRAM

calendfra
                ld      h,#00           ;calcula o end. da string
                ld      d,h             ;na memória VRAM baseada
                ld      a,(grpacy)      ;nas coordenadas passadas
                ld      l,a             ;
                or      a               ;a string está na linha 0?
                jr      z,calend1       ;Sim, vai para calend1
                ld      l,h             ;Não, calcula o end. do
                ld      b,a             ;início da linha
                ld      a,(colunas)     ;
                ld      e,a             ;
loopcal_1       add     hl,de           ;
                djnz    loopcal_1       ;
calend1         ld      a,(grpacx)      ;obtém a coord. x
                ld      e,a             ;e=coord. x
                add     hl,de           ;soma ao end. já encontrado
                ld      (bufnor+#01)    ,hl;salva no buffer das strings
                ld      (bufinv+#01)    ,hl;n rmal e invertida
                ret                     ;retorna


;a rotina posit posiciona o cursor nas coordenadas passadas
;por hl. h=x e l=y
```

```
posit
                push    af              ;salva todos os
                push    bc              ;registros afetados
                push    de              ;por esta rotina
                push    hl              ;
                ex      de,hl           ;troca o conteúdo de hl pelo
                                        ;de de
                ld      hl,#0000        ;zera hl
                ld      a,e             ;a=linha
                or      a               ;é igual a zero?
                jr      z,posit2        ;Sim, vai para posit2
                push    de              ;Não, salva as coordenadas
                ld      b,a             ;b=núm. da linha
                ld      a,(colunas)     ;a=núm. de colunas
                ld      e,a             ;e=núm. de colunas
                ld      d,#00           ;de=núm. de colunas
posit1          add     hl,de           ;hl=hl+núm. de colunas
                djnz    posit1
                pop     de              ;recupera as coordenadas
posit2          ld      a,d             ;a=coluna
                or      a               ;é igual a zero?
                jr      z,posit3        ;Sim, vai para posit3
                ld      e,a             ;Não, e=coluna
                ld      d,#00           ;de=coluna
                add     hl,de           ;hl aponta para o end. da
                                        ;VRAM
                                        ;correspondente a x1,y1
posit3          call    setvdpwt        ;prepara o VDP para escrita
                pop     hl              ;recupera os registros
                pop     de              ;
                pop     bc              ;
                pop     af              ;
                ret                     ;e retorna


;início da lista com os nomes dos novos comandos e
;endereços de chamada

lista
                defm    "REVERSE"       ;nome do comando
                defb    #00             ;fim do nome
                defw    inverte         ;end. da rotina
                defm    "CLRSCR"        ;nome do comando
                defb    #00             ;fim do nome
                defw    newcls          ;end. da rotina
                defm    "WINDOW"        ;nome do comando
                defb    #00             ;fim do nome
```

```
                defw    window          ;end. da rotina
                defm    "MENU"          ;nome do comando
                defb    #00             ;fim do nome
                defw    menu            ;end. da rotina
endlista        defb    #00             ;fim da lista


;rotinas para o acesso direto à RAM de vídeo

rdvram
                call    setvdprd        ;ajusta o VDP para leitura
                in      a,(#98)         ;lê um byte
                ret                     ;retorna


wtvram
                push    af              ;salva o dado a enviar
                call    setvdpwt        ;ajusta o VDP para escrita
                pop     af              ;recupera o dado a enviar
                out     (#98),a         ;envia
                ret                     ;retorna

setvdprd
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,rdvram1       ;Sim, vai para rdvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a         ;
rdvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde será
                and     #3f             ;lido o dado
                out     (#99),a         ;
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                ret


setvdpwt
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,wtvram1       ;Sim, vai para wtvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
```

```
                out     (#99),a
wtvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a         ;
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                ret                     ;retorna


;a rotina lelinha transporta uma linha da VRAM para a RAM

lelinha
                in      a,(#98)         ;lê o carac. da VRAM
                ld      (hl),a          ;salva-o no buffer apontado
                                        ;por hl
                inc     hl              ;incrementa o ponteiro do
                                        ;buffer
                djnz    lelinha         ;prepara a próxima leitura na
                                        ;tela
                ret                     ;retorna


;a rotina esclinha transporta uma linha da RAM para a VRAM

esclinha
                ld      a,(hl)          ;lê o carac. do buffer
                out     (#98),a         ;escreve-o na VRAM
                inc     hl              ;incrementa o ponteiro do
                                        ;buffer
                djnz    esclinha        ;prepara a próxima escrita na
                                        ;tela
                ret                     ;retorna


;área das variáveis usadas no programa

bufnor
                defb    #00             ;tamanho da string
                defw    #0000           ;posição de escrita
                defs    33              ;string

bufinv
                defb    #00             ;tamanho da string
                defw    #0000           ;posição de escrita
```

```
                defs    33              ;string

inversao        defb    #00             ;flag de campo já invertido

compstr         defb    #00             ;comprimento da string

colunas         defb    #00             ;número de colunas na tela

tabpad          defw    #0000           ;endereço da tab. de padrões

auxiliar        defw    #0000           ;ponteiro do BASIC

endvar          defw    #0000           ;end. da variável BASIC

opcao           defb    #00             ;valor da opção atual

opcaofim        defb    #00             ;valor da opção final

cooratual                               ;coordenadas atuais
yatual          defb    #00             ;y
xatual          defb    #00             ;x

bufferlinha
                defs    81              ;buffer de linha da tela

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa que implementa o comando MENU:

```
10 FOR A%=&HD000 TO &HD79E
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG25.BIN",&HD000,&HD79E
100 DATA F3,DB,A8,47,E6,F0,0F,0F,0F,0F,B0,D3,A8,E6,03,32
110 DATA 2E,D0,78,21,2D,D0,11,B1,FF,01,05,00,ED,B0,21,32
120 DATA D0,11,00,40,01,6C,07,ED,B0,D3,A8,FB,C9,F7,00,00
130 DATA 40,C9,F5,C5,D5,E5,7B,FE,02,C2,20,40,2A,AF,F6,7E
140 DATA B7,C2,16,40,23,23,23,23,DD,21,66,46,CD,B3,40,CD
150 DATA 25,40,E1,D1,C1,F1,C9,22,12,47,21,53,46,01,23,00
160 DATA ED,5B,12,47,1A,ED,B1,E2,48,40,13,1A,BE,C2,2E,40
170 DATA 23,7E,FE,00,CA,49,40,C3,38,40,C9,23,4E,23,46,F1
180 DATA C5,ED,53,12,47,EB,C9,DD,E5,DD,E5,DD,21,66,46,CD
190 DATA B3,40,DD,E1,28,1A,FE,2C,28,12,2B,DD,E5,C5,DD,21
```

```
200 DATA 0E,52,CD,B3,40,C1,DD,E1,7B,DD,77,00,DD,23,10,D9
210 DATA DD,E1,C9,3E,05,32,14,F4,18,07,3E,0D,32,14,F4,18
220 DATA 00,E1,D1,C1,F1,3A,14,F4,5F,C9,2B,AF,32,14,F4,D1
230 DATA D1,C1,F1,D1,C1,F1,DD,E1,DD,E1,F5,C5,D5,DD,21,66
240 DATA 46,C3,B3,40,C9,CD,59,01,C9,06,04,DD,21,9B,41,CD
250 DATA 55,40,22,12,47,CD,CC,40,2A,12,47,C3,98,40,DD,7E
260 DATA 00,32,B7,FC,DD,7E,01,32,B9,FC,DD,7E,02,32,0E,47
270 DATA FE,21,D2,93,41,DD,7E,03,B7,28,1C,3A,0D,47,B7,28
280 DATA 16,3A,C5,46,B7,28,10,2A,C6,46,CD,9D,46,21,C8,46
290 DATA 3A,C5,46,47,CD,BE,46,2A,12,47,3A,0E,47,B7,CA,97
300 DATA 41,DD,21,C5,46,FD,21,E9,46,DD,77,00,FD,77,00,CD
310 DATA D2,45,3A,B7,FC,5F,3A,0F,47,BB,DA,93,41,3A,B9,FC
320 DATA FE,18,D2,93,41,CD,F4,45,F3,2A,10,47,E5,11,00,07
330 DATA 19,EB,0E,E0,DD,46,00,DD,21,C8,46,FD,21,EC,46,E1
340 DATA C5,E5,D5,11,08,00,DD,46,00,19,10,FD,D1,06,08,CD
350 DATA 77,46,2F,EB,CD,7D,46,EB,23,13,10,F3,E1,C1,FD,71
360 DATA 00,0C,DD,23,FD,23,10,D8,FD,21,E9,46,FD,6E,01,FD
370 DATA 66,02,CD,9D,46,FD,46,00,21,EC,46,CD,BE,46,3E,FF
380 DATA 32,0D,47,FB,C9,F1,C3,81,40,F1,C3,98,40,00,00,00
390 DATA 00,06,02,DD,21,5C,42,CD,55,40,22,12,47,DD,7E,00
400 DATA FE,02,D2,81,40,CD,D2,45,3A,0F,47,5F,DD,7E,01,1C
410 DATA BB,D2,81,40,B7,CA,98,40,DD,7E,00,B7,C2,13,42,F3
420 DATA DD,46,01,C5,2A,B3,F3,3A,0F,47,5F,16,00,3A,B1,F3
430 DATA 47,C5,CD,85,46,3A,0F,47,D5,E5,21,1A,47,47,CD,B7
440 DATA 46,3E,20,77,E1,D1,CD,9D,46,D5,E5,21,1B,47,3A,0F
450 DATA 47,47,CD,BE,46,E1,D1,19,C1,10,D6,C1,10,C5,FB,2A
460 DATA 12,47,C3,98,40,F3,DD,46,01,C5,2A,B3,F3,3A,0F,47
470 DATA 5F,16,00,3A,B1,F3,47,C5,CD,85,46,3A,0F,47,D5,E5
480 DATA 21,1B,47,47,CD,B7,46,3E,20,21,1A,47,77,E1,D1,CD
490 DATA 9D,46,D5,E5,21,1A,47,3A,0F,47,47,CD,BE,46,E1,D1
500 DATA 19,C1,10,D3,C1,10,C2,FB,2A,12,47,C3,98,40,00,00
510 DATA DD,21,7D,43,06,05,CD,55,40,22,12,47,CD,D2,45,DD
520 DATA 21,7D,43,DD,66,00,DD,6E,01,DD,56,02,DD,5E,03,7C
530 DATA BA,D2,81,40,7D,BB,D2,81,40,7B,FE,18,D2,81,40,3A
540 DATA 0F,47,3D,BA,DA,81,40,DD,7E,04,B7,C2,A5,42,CD,1F
550 DATA 43,2A,12,47,C3,98,40,AF,32,82,43,32,83,43,22,84
560 DATA 43,ED,53,86,43,7C,82,CB,3F,3D,67,3C,3C,57,7D,83
570 DATA CB,3F,3D,6F,3C,3C,5F,CD,1F,43,CD,E6,42,CD,F5,42
580 DATA CD,1F,43,CD,04,43,CD,11,43,20,EF,2A,84,43,ED,5B
590 DATA 86,43,CD,1F,43,C3,9F,42,3A,85,43,BC,30,03,25,14
600 DATA C9,3E,01,32,82,43,C9,3A,84,43,BD,30,03,2D,1C,C9
610 DATA 3E,01,32,83,43,C9,F5,E5,21,00,10,2B,7C,B5,20,FB
620 DATA E1,F1,C9,3A,82,43,FE,01,28,01,C9,3A,83,43,FE,01
630 DATA C9,C5,D5,E5,CD,2B,46,7A,94,3D,47,C5,3E,18,D3,98
640 DATA E3,E3,3E,17,D3,98,E3,E3,10,F8,3E,19,D3,98,C1,6B
650 DATA CD,2B,46,3E,1A,D3,98,E3,E3,3E,17,D3,98,E3,E3,10
660 DATA F8,3E,1B,D3,98,E1,D1,D5,E5,7B,95,3D,47,7A,94,3D
```

```
670 DATA 4F,2C,CD,2B,46,3E,16,D3,98,C5,41,3E,20,D3,98,E3
680 DATA E3,10,F8,3E,16,D3,98,2C,C1,10,E7,E1,D1,C1,C9,00
690 DATA 00,00,00,00,00,00,00,00,00,00,DD,21,83,45,06,06
700 DATA CD,55,40,2B,DD,21,66,46,CD,B3,40,FE,2C,C2,20,40
710 DATA DD,21,66,46,CD,B3,40,CA,20,40,DD,21,A4,5E,CD,B3
720 DATA 40,ED,53,14,47,3A,63,F6,FE,02,C2,88,40,22,12,47
730 DATA DD,21,9B,41,FD,21,83,45,FD,7E,02,FD,BE,00,DA,81
740 DATA 40,FD,7E,03,FD,BE,01,DA,81,40,CD,D2,45,3A,0F,47
750 DATA 3D,FD,BE,00,DA,81,40,FD,BE,02,DA,81,40,3E,18,FD
760 DATA BE,01,DA,81,40,FD,BE,03,DA,81,40,FD,66,00,FD,6E
770 DATA 01,FD,56,02,FD,5E,03,FD,7E,04,DD,77,02,3E,01,DD
780 DATA 77,03,22,18,47,AF,32,16,47,32,0D,47,7C,BA,CA,D3
790 DATA 44,7D,BB,C2,20,40,3A,87,45,5F,3A,88,45,83,5F,3A
800 DATA 83,45,4F,3A,85,45,06,00,93,38,04,04,B9,20,F9,78
810 DATA 32,17,47,DD,21,9B,41,2A,18,47,7C,DD,77,00,7D,DD
820 DATA 77,01,CD,CC,40,CD,89,45,FE,03,CA,67,44,FE,07,CA
830 DATA 9B,44,FE,FF,CA,70,45,18,DA,3A,88,45,5F,3A,19,47
840 DATA 83,5F,3A,87,45,83,5F,3A,85,45,BB,38,0E,7B,32,19
850 DATA 47,3A,16,47,3C,32,16,47,C3,41,44,3A,83,45,67,3A
860 DATA 84,45,6F,22,18,47,AF,32,16,47,C3,41,44,3A,88,45
870 DATA 5F,3A,19,47,4F,3A,83,45,B9,28,16,79,93,5F,3A,87
880 DATA 45,4F,7B,91,32,19,47,3A,16,47,3D,32,16,47,C3,41
890 DATA 44,3A,85,45,67,3A,86,45,6F,22,18,47,3A,17,47,32
900 DATA 16,47,C3,41,44,3A,88,45,5F,3A,84,45,4F,3A,86,45
910 DATA 06,00,93,04,B9,20,FB,78,32,17,47,DD,21,9B,41,2A
920 DATA 18,47,7C,DD,77,00,7D,DD,77,01,CD,CC,40,CD,89,45
930 DATA FE,05,CA,10,45,FE,01,CA,40,45,FE,FF,CA,70,45,C3
940 DATA E9,44,3A,88,45,5F,3A,18,47,83,5F,3A,86,45,BB,DA
950 DATA 2E,45,7B,32,18,47,3A,16,47,3C,32,16,47,C3,E9,44
960 DATA 3A,83,45,67,3A,84,45,6F,22,18,47,AF,32,16,47,C3
970 DATA E9,44,3A,88,45,5F,3A,84,45,4F,3A,18,47,B9,28,0E
980 DATA 93,32,18,47,3A,16,47,3D,32,16,47,C3,E9,44,3A,85
990 DATA 45,67,3A,86,45,6F,22,18,47,3A,17,47,32,16,47,C3
1000 DATA E9,44,2A,14,47,3A,16,47,77,23,36,00,CD,56,01,2A
1010 DATA 12,47,C3,98,40,00,00,00,00,00,00,DD,E5,FD,E5,C5
1020 DATA D5,E5,3E,00,CD,D5,00,B7,20,28,3E,01,CD,D5,00,B7
1030 DATA 20,20,3E,02,CD,D5,00,B7,20,18,3E,00,CD,D8,00,B7
1040 DATA 20,10,3E,01,CD,D8,00,B7,20,08,3E,02,CD,D8,00,B7
1050 DATA 28,D0,F5,21,00,80,2B,7C,B5,20,FB,F1,E1,D1,C1,FD
1060 DATA E1,DD,E1,C9,08,D9,2A,B7,F3,3A,2D,00,B7,28,0C,3A
1070 DATA B0,F3,FE,29,38,05,3E,50,29,18,02,3E,28,32,0F,47
1080 DATA 22,10,47,D9,08,C9,F5,C5,D5,E5,CD,0C,46,CD,85,46
1090 DATA DD,46,00,21,C8,46,CD,B7,46,E1,D1,C1,F1,C9,26,00
1100 DATA 54,3A,B9,FC,6F,B7,28,09,6C,47,3A,0F,47,5F,19,10
1110 DATA FD,3A,B7,FC,5F,19,22,C6,46,22,EA,46,C9,F5,C5,D5
1120 DATA E5,EB,21,00,00,7B,B7,28,0C,D5,47,3A,0F,47,5F,16
1130 DATA 00,19,10,FD,D1,7A,B7,28,04,5F,16,00,19,CD,9D,46
```

```
1140 DATA E1,D1,C1,F1,C9,52,45,56,45,52,53,45,00,B7,40,43
1150 DATA 4C,52,53,43,52,00,9F,41,57,49,4E,44,4F,57,00,5E
1160 DATA 42,4D,45,4E,55,00,88,43,00,CD,85,46,DB,98,C9,F5
1170 DATA CD,9D,46,F1,D3,98,C9,3A,2D,00,B7,28,07,AF,D3,99
1180 DATA 3E,8E,D3,99,7D,D3,99,7C,E6,3F,D3,99,E3,E3,C9,3A
1190 DATA 2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99,7C
1200 DATA E6,3F,F6,40,D3,99,E3,E3,C9,DB,98,77,23,10,FA,C9
1210 DATA 7E,D3,98,23,10,FA,C9,00,00,00,00,00,00,00,00,00
1220 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1230 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1240 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1250 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1260 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1270 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1280 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1290 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1300 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1310 DATA 00,00,00,00,00,00,00,00,00,00,00,00,6C,69
```

Listagem do programa de teste:

```
1000 CLS:KEYOFF
1010 REM *** CARREGA OS PROGRAMAS NECESSÁRIOS ***
1020 BLOAD"PROG25.BIN",R
1030 BLOAD"PROG6.BIN"
1040 BLOAD"PROG7.BIN"
1050 DEFUSR0=&HD000:DEFUSR1=&HD200:RESTORE 1700
1060 REM *** CONSTRóI O MENU PRINCIPAL ***
1070 FOR A%=1 TO 4
1080 READ A$(A%)
1090 NEXT A%
1100 FOR A%=1 TO 4
1110 LOCATE (A%-1)*7+1,0:PRINT A$(A%)
1120 NEXT A%
1130 REM *** TEXTO EXPLICATIVO ***
1140 S$(1)="SELECIONE A OPÇÃO DESEJADA"
1150 S$(2)="USANDO AS TECLAS DO CURSOR"
1160 IF (PEEK(&HF3B0))>40 THEN CT%=80 ELSE CT%=40
1170 X1=(CT%-LEN(S$(1)))/2-1:X2=X1+LEN(S$(1))+1:Y1=8:Y2=12
1180 WINDOW X1,Y1,X2,Y2,0
1190 LOCATE X1+1,Y1+1:PRINT S$(1)
1200 LOCATE X1+1,Y1+3:PRINT S$(2)
1210 REM *** LOOP DE LEITURA DO MENU PRINCIPAL
1220 MENU 1,0,22,0,4,3,A%
1230 REVERSE ,,0,1:REM *** APAGA A OPÇÃO ***
```

```
1240 ON A%+1 GOSUB 1280,1400,1520,1630
1250 Z%=USR1(0):REM *** RECUPERA A TELA ***
1260 GOTO 1220
1270 REM *** MENU DA PRIMEIRA OPÇÃO ***
1280 RESTORE 1710:CP%=0
1290 FOR B%=1 TO 6
1300 READ B$(B%)
1310 IF LEN(B$(B%))>CP% THEN CP%=LEN(B$(B%))
1320 NEXT B%
1330 GOSUB 1650:REM *** DESENHA A JANELA ***
1340 FOR B%=1 TO 6
1350 LOCATE X1+1,Y1+B%:PRINT B$(B%)
1360 NEXT B%
1370 MENU X1+1,Y1+1,X1+1,Y2-1,CP%,1,B%
1380 RETURN
1390 REM *** MENU DA SEGUNDA OPÇÃO ***
1400 RESTORE 1730:CP%=0
1410 FOR B%=1 TO 4
1420 READ B$(B%)
1430 IF LEN(B$(B%))>CP% THEN CP%=LEN(B$(B%))
1440 NEXT B%
1450 GOSUB 1650:REM *** DESENHA A JANELA ***
1460 FOR B%=1 TO 4
1470 LOCATE X1+1,Y1+B%:PRINT B$(B%)
1480 NEXT B%
1490 MENU X1+1,Y1+1,X1+1,Y2-1,CP%,1,B%
1500 RETURN
1510 REM *** MENU DA TERCEIRA OPÇÃO ***
1520 RESTORE 1750:CP%=0
1530 FOR B%=1 TO 4
1540 READ B$(B%)
1550 IF LEN(B$(B%))>CP% THEN CP%=LEN(B$(B%))
1560 NEXT B%
1570 GOSUB 1650:REM *** DESENHA A JANELA ***
1580 FOR B%=1 TO 4
1590 LOCATE X1+1,Y1+B%:PRINT B$(B%)
1600 NEXT B%
1610 MENU X1+1,Y1+1,X1+1,Y2-1,CP%,1,B%
1620 RETURN
1630 END
1640 REM *** ROTINA PARA DESENHAR A JANELA ***
1650 X1=A%*7:X2=X1+CP%+2:Y1=1:Y2=Y1+B%
1660 Z%=USR0(0):REM salva a tela atual
1670 WINDOW X1,Y1,X2,Y2,0
1680 RETURN
1690 REM *** DADOS USADOS NOS MENUS ***
1700 DATA "FILE","EDIT","COMP","EXIT"
```

```
1710 DATA "Copy","View","Delete"
1720 DATA "Rename","Print","Save"
1730 DATA "Insert","Delete"
1740 DATA "Undo","Copy"
1750 DATA "To memory","To disk"
1760 DATA "Make .EXE","Make .COM"
```

Antes de passar às análises das listagens acima, gostaria de observar que a rotina do comando **REVERSE** sofreu algumas modificações para se adaptar ao comando **MENU**, portanto, fique atento a esta nova listagem.

Ao executar o programa de teste, você terá um pequeno exemplo (tenho certeza que você será bem mais criativo) de como e quando usar o comando MENU. Observe que, para tornar o efeito visual muito mais interessante, usei o comando WINDOW em conjunto com as rotinas do Capítulo 1 que salvam e recuperam uma tela de texto. O resultado é bem interessante, não? Existem, porém, algumas restrições no uso do comando **MENU**; são elas:

1.Não deve existir nenhum campo invertido no momento da execução do comando MENU. Para se certificar que tal condição está satisfeita, utilize o comando

REVERSE ,,0,1

antes do comando MENU.

2.Quando o menu estiver na horizontal, você deverá ter muito cuidado em fornecer ao comando MENU opções com espaçamentos fixos entre si, ou seja, o deslocamento entre uma opção e outra deve ser sempre o mesmo. Caso contrário, o sistema produzirá um erro de chamada ilegal.

3.Certifique-se que duas coordenadas (x1 e x2, se o movimento for na vertical, ou y1 e y2, se o movimento for na horizontal) sejam iguais, pois a rotina detecta a direção do movimento baseada nessa igualdade. Caso contrário, o sistema produzirá uma mensagem de erro de sintaxe.

Fora as observações acima, bastará acompanhar os comentários na listagem do código-fonte para entender o funcionamento de cada uma das rotinas apresentadas. Como você pode perceber, existem muitos comandos que poderão ser implementados usando-se o método do desvio do vetor de erro. Acredito que, uma vez com as ferramentas fornecidas neste capítulo e no Capítulo 2, você será capaz de implementar os comandos que achar mais convenientes.

Com as listagens acima, encerramos o Capítulo 3 e passamos à aplicação prática de um assunto que não abordei nos meus livros anteriores, mas que, mesmo assim, recebi diversas cartas trazendo dúvidas a respeito. No Capítulo 4, vamos explorar o funcionamento da instrução **CALL** do **BASIC**.

BIBLIOGRAFIA RECOMENDADA

INTRODUÇÃO À LINGUAGEM DE MÁQUINA PARA MSX - Eduardo Alberto Barbosa - Rio de Janeiro - CIÊNCIA MODERNA COMPUTAÇÃO LTDA.

Capítulo 4

A INSTRUÇÃO CALL

Para entender como funciona a instrução **CALL** do BASIC, precisamos de alguns conhecimentos de base, entre eles o funcionamento do sistema de cartuchos no MSX.

Embora a Microsoft tenha divulgado no exterior um catálogo com todas as informações sobre o sistema MSX (inclusive de hardware), nunca vi nenhum lançamento nacional com tais informações. Já que você não tem acesso a essa biblioteca, vou apresentar um panorama geral sobre como o MSX interpreta o sistema de cartuchos.

A INICIALIZAÇÃO DO MSX

Quando você liga o MSX, ativa uma série de rotinas complexas que têm como função inicializá-lo, tornando-o pronto para o uso. O que fazem exatamente essas rotinas? Em primeiro lugar, o MSX faz uma inicialização de todos os periféricos "internos", ou seja, inicializa o vídeo, o processador de sons e parte do ppi (o chip que controla o teclado, o motor do gravador cassete, a paginação da memória, etc). Logo depois, sai em busca de um bloco contínuo de 16Kb de memória RAM que se estenda do endereço #8000 a #BFFF. O curioso é que essa busca se realiza de cima para baixo, ou seja, do endereço #BFFF ao #8000. Achado o bloco de memória RAM de #8000 a #BFFF (página 2), o MSX inicia a busca de um bloco de memória RAM que se estenda do endereço #C000 a #FFFF (página 3). Achados os blocos, o MSX parte para a busca de um bloco de 32Kb contínuos que se estenda do endereço #8000 a #FFFF. Terminada esta etapa, é feito um teste para habilitar o bloco de 32Kb de memória RAM que se encontre no slot mais próximo ao slot 0. Após este passo, o MSX termina a inicialização definitiva do ppi e passa para a inicialização das variáveis do sistema. Em seguida à inicialização das variáveis do sistema, o MSX inicia a procura de cartuchos de expansão (jogos, interface de drive, programas extensivos em ROM, etc.) e, se for o caso (somente cartuchos de jogos), desvia a execução para um deles.

O SISTEMA DE CARTUCHOS

Já sabemos que após a habilitação dos 32Kb contínuos de memória RAM (páginas 2 e 3) e a inicialização da área das variáveis do sistema, o MSX inicia a busca de cartuchos de expansão. Mas, como isso é feito? Acontece que os cartuchos de expansão só podem ocupar a página 1 ou 2, ou seja, a partir do endereço #4000 (jogos, manipuladores de comandos e ma-

nipuladores de dispositivos) ou a partir do endereço #8000 (cartuchos com programas em BASIC). Assim sendo, o que o MSX tem de verificar são os dois primeiros bytes do endereço #4000 e #8000 de todos os slots e subslots (no caso de slots expandidos). Se esses bytes forem iguais a #41 e #42, respectivamente, o MSX reconhece que ali existe um cartucho de expansão. Curioso salientar que este também é o método usado pelo IBM-PC no reconhecimento de extensões da ROM, só que os bytes de identificação mudam de #41 e #42 para #55 e #AA. Após este reconhecimento, o MSX parte para uma outra verificação, para poder se ajustar ao tipo do cartucho. Fundamentalmente, existem quatro tipos de cartucho, quais sejam:

1.Cartuchos com jogos;

2.Cartuchos com manipuladores de comandos ativados por intermédio da instrução **CALL** do BASIC;

3.Cartuchos com manipuladores de dispositivos, como a RS232, ativados por intermédio do comando **OPEN** e associados (**CLOSE,** etc.) do BASIC, e;

4.Cartuchos com programas em BASIC.

Os três primeiros tipos ocupam os 16Kb da página 1 (#4000 a #7FFF) do slot ou subslot onde estão localizados; já o quarto tipo ocupa os 16Kb da página 2 (#8000 a #BFFF) do slot onde está localizado. Um exemplo do último tipo é o cartucho de demonstração que acompanha o modelo Expert. "Mas, Eduardo, como é que o MSX reconhece cada um dos tipos de cartucho?" Para entender o reconhecimento, veja a Figura 4.1 abaixo.

INÍCIO + #0000	**IDENTIFICAÇÃO** = #4142	
+ #0002	**INIT**	= Endereço inicial de execução do jogo
+ #0004	**STATEMENT**	= Endereço inicial de execução de um manipulador de comandos
+ #0006	**DEVICE**	= Endereço inicial de execução de um manipulador de dispositivos
+ #0008	**TEXT**	= Endereço inicial do programa em BASIC
+ #000A	**RESERVADO PARA EXPANSÕES FUTURAS**	
+ #0010	**INÍCIO DOS PROGRAMAS**	

Onde INÍCIO ***pode ser igual a #4000 ou a #8000, dependendo do tipo do cartucho***

Figura 4.1: Tabela dos parâmetros dos cartuchos

Dessa forma, se, por exemplo, o MSX encontrar na página 1 (#4000 a #7FFF) do slot ou subslot que estiver examinando a seguinte seqüência de bytes:

41 42 10 40 00 00 00 00 00 00

saberá que ali se encontra um cartucho de jogo para o qual deverá ser transferida imediatamente a execução. Note que, neste exemplo,a execução seria transferida para o endereço #4010. Por outro lado, se o MSX encontrar uma seqüência do tipo

41 42 00 00 08 40 00 00 00 00

saberá que nesse slot/subslot, em particular, existe um manipulador de comandos ativado por intermédio da instrução **CALL** do BASIC. Neste caso, a execução não é transferida para o endereço #4008, como mostra o exemplo acima. Ao invés disso,o MSX inicializa as variáveis do sistema que indicarão ao BASIC que nesse slot/subslot existe um manipulador de comandos a ser ativado pela instrução **CALL**. Já se o MSX encontrar a seguinte seqüência de bytes:

41 42 00 00 00 00 10 40 00 00

saberá que na página 1 do slot/subslot que está sendo analisado encontra-se um manipulador de dispositivos (como uma RS232, por exemplo) a ser ativado pelas instruções do BASIC. Neste caso, a execução também não é transferida para o endereço #4010, como mostra a seqüência de bytes acima. O que ocorre, na realidade, é uma inicialização das variáveis do sistema, que por sua vez informarão ao BASIC que nesse slot/subslot existe um manipulador de dispositivos. Por último, se o MSX encontrar uma seqüência do tipo

41 42 00 00 00 00 00 00 10 80

saberá que na página 2 (#8000 a #BFFF) existe um programa em BASIC com início a partir do endereço #8010. Neste caso, o MSX promove um desvio das variáveis do sistema relacionadas às tabelas usadas pelos programas em BASIC (início do texto do programa, início da área de variáveis, etc.) e desvia a execução para o comando **RUN** na BIOS do BASIC, o que por sua vez resultará na execução do programa BASIC gravado no cartucho. Observe que a presença do endereço #0000 em INIT, STATEMENT, DEVICE ou TEXT indica a ausência de tal tipo de cartucho no slot/subslot em questão. Observe também que podemos ter dois tipos de cartucho num mesmo slot/subslot. Veja, por exemplo, a seqüência

41 42 10 40 40 40 00 00 00 00

Ela indica que existe uma rotina a ser ativada imediatamente no endereço #4010 (talvez para promover alguma inicialização) e, ao mesmo tempo, existe um manipulador de comandos a partir do endereço #4040. Neste caso, a rotina que se inicia em #4010 deverá terminar com uma instrução **RET** (em assembly) para permitir uma volta à inicialização do MSX, de modo que este possa descobrir que nesse slot/subslot também existe um manipulador de comandos.

Para nós, só interessa estudar o manipulador de comandos (STATEMENT). Vejamos os motivos:

1.Os cartuchos para jogos (INIT) só têm sentido se você for programar jogos para serem utili-

zados em cartuchos. Ora, isto implica ter acesso a um gravador de EPROMs, que vem a ser um equipamento que pouquíssimos usuários possuem.

2.Os cartuchos para manipuladores de dispositivos (DEVICE) só têm sentido se tivermos conectado ao nosso MSX algum tipo de periférico, como a RS232, por exemplo. Este também não é o caso da esmagadora maioria dos usuários do MSX.

3.Os cartuchos com programas em BASIC também só têm sentido para programas de demonstração ou aplicativos em BASIC programados para serem comercializados em cartuchos. Mais uma vez, surge a necessidade de se possuir um gravador de EPROMs.

O MANIPULADOR DE COMANDOS (STATEMENT)

Quando o interpretador BASIC encontra uma sentença do tipo

CALL<*Nome do Comando*> [<*Lista de argumentos*>]

transfere para a variável do sistema **PROCNM** o ***Nome do Comando*** e promove uma consulta à tabela contida na variável do sistema **SLTATR** para encontrar os slots/subslots que possuam um manipulador de comandos. Logo após, o sistema promove um desvio para as rotinas em cada um desses slots/subslots até que uma delas reconheça o ***Nome do Comando*** e passe a executar as instruções pertinentes.

Antes, porém, para podermos usar o manipulador de comandos, precisamos seguir algumas regras:

1.O par de registros HL deve ser preservado, já que é o apontador usado pelo interpretador BASIC. Como vimos no Capítulo 2, o interpretador BASIC ***varre*** o programa em BASIC usando o par HL.

2.Por ser um manipulador de comandos, a rotina e os bytes de identificação (#41, #42, etc.) devem se localizar na página 1 (do endereço #4000 ao #7FFF) de qualquer slot/subslot.

3.Respeitar a sintaxe de chamada vista no início deste tópico.

4.Ao ser ativada, a rotina do manipulador de comandos deve, primeiramente, comparar o nome passado pelo interpretador BASIC com o nome da própria rotina. O nome passado pelo interpretador BASIC é colocado em **PROCNM** e apresenta um comprimento de, no máximo, 15 caracteres. O fim do nome é assinalado com o caractere **NUL** (#00)

5.Se o nome da rotina não coincidir com o nome colocado em **PROCNM**, a rotina deverá voltar ao BASIC recuperando o par HL salvo em **1** e setando a flag de carry para indicar que o nome não coincidiu. Se todos os manipuladores analisados retornarem com a flag de carry setada, o interpretador BASIC emitirá um erro de sintaxe.

6.Se o teste de comparação resultar em verdadeiro, a rotina deverá ser executada e, no final, retornar recuperando o par de registros HL salvo em 1 e resetando a flag de carry (a instrução XOR A em assembly realiza tal função), para que o sistema não emita uma mensagem de erro de sintaxe.

Como consolo, resta saber que todos os registros podem ser alterados, com exceção do ponteiro HL e do ponteiro SP da pilha, ou seja, todos os PUSHs que fizermos deverão ser correspondidos por um número igual de POPs.

A partir de agora, já temos quase todas as informações necessárias para poder implementar o nosso comando **RENEW** a ser ativado através da instrução **CALL** do BASIC, e que tem como função recuperar programas em BASIC apagados acidentalmente.

Antes, porém, precisamos entender o gerenciamento de algumas variáveis do sistema relacionadas com o sistema de cartucho.

VARIÁVEIS DO SISTEMA E O SISTEMA DE CARTUCHOS

Apesar de existirem diversas variáveis do sistema para o gerenciamento do mecanismo dos slots, vamos nos preocupar apenas com aquelas que dizem respeito ao manipulador de comandos (STATEMENT). Como você já sabe, durante a inicialização (alguns preferem o termo **RESET**) do MSX, é feita uma busca de cartuchos de expansão para poder inicializar as variáveis do sistema correspondentes. Temos, então, duas alternativas para implementar o nosso comando **RENEW** a ser ativado pela instrução **CALL** do BASIC:

1.Deslocar a nossa rotina do endereço de carregamento na RAM do BASIC para a página 1 do slot que contenha os 64Kb de RAM e provocar um reset através da instrução **JUMP #0000** em assembly. No nosso caso, isso significaria deslocar a rotina do endereço #D000 para o endereço #4000 (habilitando antes a página 1 da RAM, é claro) e, logo em seguida, provocar um reset usando a instrução **JUMP #0000**, para que na nova inicialização o sistema reconhecesse o nosso comando. Mas, eu acho que esta não é uma alternativa elegante. Suponha que o usuário tenha apagado acidentalmente algum programa em BASIC e deseje recuperá-lo. Se a instalação do nosso comando provocar um reset, muito provavelmente tal programa jamais será recuperado.

2.Deslocar a nossa rotina do endereço de carregamento na RAM do BASIC para a página 1 do slot que contenha os 64Kb de RAM e promover a mesma inicialização das variáveis do sistema que seria feita pelo reset do sistema. Francamente, acho este método muito mais simpático que o primeiro.

Vamos, então, passar ao estudo das variáveis do sistema envolvidas na utilização do manipulador de comandos. As variáveis envolvidas são:

SLTATR que se inicia no endereço #FCC9 e ocupa 64 bytes. Esta variável contém um mapa completo de todos os cartuchos de expansão em todos os slots e subslots. Como podemos ter até 16 slots (4 subslots por cada um dos 4 slots principais) e como cada slot possui 4 páginas de 16Kb (cada slot só pode ter 64Kb de memória - limitação imposta pelo processador de 8 bits), chegamos aos 16*4=64 bytes ocupados por esta tabela.

PROCNM que se inicia no endereço #FD89 e ocupa 16 bytes, sendo 15 bytes para o armazenamento temporário do nome do comando ativado pela instrução CALL e 1 byte sempre igual a #00 para indicar o término do nome.

Das duas variáveis acima, a mais complexa é a **SLTATR**. Vejamos a sua estrutura:

```
FCC9H  SLTATR:     DEFS   4      ;SP0,SS0
FCCDH              DEFS   4      ;SP0,SS1
FCD1H              DEFS   4      ;SP0,SS2
FCD5H              DEFS   4      ;SP0,SS3

FCD9H              DEFS   4      ;SP1,SS0
FCDDH              DEFS   4      ;SP1,SS1
FCE1M              DEFS   4      ;FS1,SS2
FCE5H              DEFS   4      ;SP1,SS3

FCE9H              DEFS   4      ;SP2,SS0
FCEDH              DEFS   4      ;SP2,SS1
FCF1H              DEFS   4      ;SP2,SS2
FCF5H              DEFS   4      ;SP2,SS3

FCF9H              DEFS   4      ;SP3,SS0
FCFDH              DEFS   4      ;SP3,SS1
FD01H              DEFS   4      ;SP3,SS2
FD05H              DEFS   4      ;SP3,SS3
```

Onde SP é a abreviação para Slot Principal e SS é a abreviação para Slot Secundário

Tabela 4.1: A variável do sistema SLTATR

"Mas, o que os endereços acima significam?" Bem, não é nada complicado. Vejamos um exemplo: suponha que o seu micro seja um Expert 1.0 e que, portanto, apresenta os 64Kb de RAM no slot 2. Agora, vamos também supor que desejemos colocar o nosso manipulador de comandos na página 1 do slot principal. Consultando a Tabela 4.1, chegamos à conlusão de que o endereço inicial do slot 2 (SP2) é #FCE9, mas, como queremos modificar a página 1, devemos apontar para o endereço seguinte, ou seja, para o endereço #FCEA. Note que, na ausência de slots expandidos, a abreviação SS0 se aplica ao slot principal. Assim sendo, fica claro que o mnemônico **DEFS 4** reserva na verdade 4 bytes, sendo um para cada uma das quatro páginas (0 a 3) possíveis. Mas, o que devemos colocar nessas posições de memória para informar ao sistema que numa determinada página de um determinado slot/subslot existe um manipulador de comandos? Para isto, devemos seguir a seguinte codificação em bits colocada em cada um dos 64 bytes da tabela SLTATR:

BIT	SIGNIFICADO
7	Programa em BASIC
6	Manipulador de dispositivos
5	Manipulador de comandos
4	Não usado
3	Não usado
2	Não usado
1	Não usado
0	Não usado

Como já sabemos, só tem sentido setar (colocar em 1) o bit 7 na página 2 (#8000 a #BFFF) do slot/subslot em questão, da mesma forma que só tem sentido setar os bits 6 e 5 na página 1 (#4000 a #7FFF) do slot/subslot em questão. Desta forma, fica claro que os valores correspondentes às páginas 0 e 3 da tabela serão sempre iguais a zero (sem nenhum bit setado). Terminada a teoria, vamos à aplicação prática destes princípios: a implementação do comando **RENEW**.

O COMANDO RENEW

Como já afirmei, este comando será implementado para ser ativado por intermédio da instrução **CALL** do BASIC. A sua função é recuperar programas em BASIC apagados acidentalmente. Antes, porém, vamos a uma pequena explicação do algoritmo usado para tal recuperação. Você deve estar lembrado que, no Capítulo 2, vimos que o armazenamento de uma linha em BASIC é feita da seguinte forma:

```
INÍCIO  + #0000 Endereço da próxima linha
        + #0002 Número da linha
        + #0004 Primeiro comando da linha
                .....
                .....
                Fim da linha marcado pelo byte #00
```

Ao entrarmos com o comando **NEW**, o sistema simplesmente reinicializa três ponteiros que se situam na área das variáveis do sistema, e preenche com zeros os primeiros 2 bytes da primeira linha, ou seja, perdemos o indicativo do endereço inicial da segunda linha. Assim sendo, o que temos a fazer é procurar o endereço do fim da primeira linha (indicado pelo byte #00), incrementar o endereço encontrado para que ele aponte para o início da segunda linha e, por fim, colocar tal endereço nos dois primeiros bytes da primeira linha. Feito isto, precisamos inicializar os três ponteiros na área das variáveis do sistema. A função desses ponteiros é indicar o início da área destinada às variáveis do programa BASIC; são eles:

VARTAB que se inicia no endereço #F6C2 e que tem como função apontar para o primeiro byte da área de armazenamento de variáveis.

ARYTAB que se inicia no endereço #F6C4 e que tem como função apontar para o primeiro byte da área de armazenamento de arrays.

STREND que se inicia no endereço #F6C6 e que tem como função apontar para o primeiro byte após a área de armazenamento de arrays.

"Mas, quais são os valores a serem colocados em cada um desses ponteiros?" Simples. Todos eles são gerenciados pelo interpretador BASIC à medida que o programa em BASIC vai sendo executado. Sendo assim, basta fazer com que todos esses ponteiros apontem para o fim do programa em BASIC e, depois, o comando **RUN** se encarregará de inicializá-los com os valores apropriados. Ficamos, então, na dependência de achar o final do programa em BASIC, não é? Você vai ver que é muito simples achar tal endereço. No Capítulo 2, vimos que um programa em BASIC termina com uma seqüência de três bytes iguais a #00, sendo que o primeiro byte corresponde ao fim da última linha e os dois bytes seguintes ao início da linha seguinte, que não existe. O que eu sugiro é que obtenhamos no início de cada linha o endereço da linha seguinte, para verificarmos se o conteúdo desse endereço é igual a #0000 (é claro que estaremos analisando também o conteúdo do endereço seguinte, pois #0000 é uma quantidade de 16 bits). Se esta comparação for verdadeira, achamos o fim do programa em BASIC; caso contrário, tomamos o conteúdo do endereço analisado, que nada mais é do que o endereço para a linha seguinte, e repetimos a comparação até encontrar o valor #0000. Como você pode comprovar, não existe qualquer mistério. Vamos, então, às listagens do programa que implementa o comando **RENEW**.

Listagem em assembly Z-80 do código-fonte do programa que implementa o comando RENEW:

```
;programa que implementa o comando RENEW
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG26.BIN=PROG26.GEN
;
;onde PROG26.GEN é o nome do arquivo-texto com esta
;listagem

txttab          equu    #f676
vartab          equ     #f6c2
arytab          equ     #f6c4
strend          equ     #f6c6
sltatr          equ     #fcc9
procnm          equ     #fd89
```

```
                defb    #fe                 ;simula em CP/M
                defw    inicio              ;o cabecalho de
                defw    fim-renew+rotina+#01
                                            ;um arquivo
                defw    inicio              ;.BIN


                org     #d000

inicio
                di                          ;desabilita as interrupções
                in      a,(#a8)             ;lê a atual configuração dos
                ld      b,a                 ;slots e prepara para ativar
                and     #f0                 ;a página 1 do slot da RAM
                rrca                        ;
                rrca                        ;
                rrca                        ;
                rrca                        ;
                or      b                   ;
                out     (#a8),a             ;ativa as páginas 0,1,2 e 3
                and     #03                 ;em RAM e descobre o slot
                                            ;onde se encontram os 64Kb
                                            ;de RAM
                ld      hl,sltatr           ;hl aponta para sltatr
                ld      de,16               ;de=incremento
                or      a                   ;slot 0?
                jr      z,inicio_1          ;Sim, vai para inicio_1
                ld      b,a                 ;Não, calcula endereço
inicio_0        add     hl,de               ;da área de dados do
                djnz    inicio_0            ;slot em questão
inicio_1        xor     a                   ;zera o acumulador
                set     5,a                 ;indica manipulador
                                            ;de comando (CALL)
                inc     hl                  ;hl aponta para a área
                                            ;de dados da página 1
                                            ;do slot em questão
                ld      (hl),a              ;indica ao sistema que
                                            ;a página 1 do slot da
                                            ;RAM contém uma rotina
                                            ;a ser ativada por CALL
                ld      hl,rotina           ;transfere a rotina para a
                ld      de,renew            ;página 1 da RAM
                ld      bc,fim-renew+#01;
                ldir                        ;
                in      a,(#a8)             ;lê a configuração atual
                and     %11110000           ;ativa os 32Kb de ROM
                out     (#a8),a             ;habilita a config. original
```

```
                ei                          ;habilita as interrupções
                ret                         ;retorna ao BASIC


rotina
                org     #4000

renew
                defb    #41,#42             ;indicativo de expansão
                defw    #0000               ;não é jogo
                defw    inirenew            ;end. de entrada do
                                            ;manipulador de comandos
                defs    10                  ;zera os 10 bytes restantes

inirenew
                ld      (ptrbasic),hl       ;salva o ptr. do BASIC
                ld      hl,procnm           ;hl aponta para procnm
                ld      de,comando          ;de para o nome do comando
renew_0         ld      a,(de)              ;loop para comparação
                                            ;dos nomes
                or      a                   ;fim do nome?
                jr      z,renew_1           ;Sim, ativa a rotina
                cp      (hl)                ;Não, continua a comparação
                jr      nz,voltarenew       ;Falhou. Volta com erro
                inc     hl                  ;Não falhou. Continua a
                inc     de                  ;comparação
                jr      renew_0             ;

voltarenew
                ld      hl,(ptrbasic)       ;recupera o ptr. do BASIC
                scf                         ;seta a flag de carry
                                            ;para indicar erro
                ret                         ;volta ao BASIC

renew_1
                ld      hl,(txttab)         ;hl aponta para o início
                                            ;do programa em BASIC
                inc     hl                  ;
                inc     hl                  ;
                inc     hl                  ;
                inc     hl                  ;
renew_10        ld      a,(hl)              ;loop para procurar o
                or      a                   ;final da primeira linha
                jr      z,renew_11          ;Encontrou? Vai para
                                            ;renew_11
                inc     hl                  ;Não, continua a procura
                jr      renew_10            ;
```

```
renew_11        inc     hl              ;hl aponta para o início
                                        ;da segunda linha
                ex      de,hl           ;de=hl
                ld      hl,(txttab)     ;recupera o ponteiro
                ld      (hl),e          ;para a segunda linha
                inc     hl              ;
                ld      (hl),d          ;

renew_2
                ex      de,hl           ;hl=de
                ld      e,(hl)          ;de=início da próxima
                inc     hl              ;linha
                ld      d,(hl)          ;
                ld      a,d             ;o número da linha
                or      e               ;é zero?
                jr      nz,renew_2      ;Não, continua a procura

achoufim
                inc     hl              ;Sim, achou o final do
                                        ;programa em BASIC
                                        ;ajusta os vetores
                ld      (vartab),hl     ;necessários
                ld      (arytab),hl     ;
                ld      (strend),hl     ;
                ld      hl,(ptrbasic)   ;recupera o ptr. do BASIC
                xor     a               ;zera a flag de carry
                ret                     ;volta ao BASIC sem
                                        ;erro

comando         defm    "RENEW"
                defb    #00

ptrbasic        defw    #0000

fim             equ     $
```

Listagem do código-objeto do programa que implementa o comando RENEW:

```
10 FOR A%=&HD000 TO &HD093
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG26.BIN",&HD000,&HD093
100 DATA F3,DB,A8,47,E6,F0,0F,0F,0F,0F,B0,D3,A8,E6,03,21
110 DATA C9,FC,11,10,00,B7,28,04,47,19,10,FD,AF,CB,EF,23
120 DATA 77,21,34,D0,11,00,40,01,5F,00,ED,B0,DB,A8,E6,F0
```

```
130 DATA D3,A8,FB,C9,41,42,00,00,10,40,00,00,00,00,00,00
140 DATA 00,00,00,00,22,5C,40,21,89,FD,11,56,40,1A,B7,28
150 DATA 0C,BE,20,04,23,13,18,F5,2A,5C,40,37,C9,2A,76,F6
160 DATA 23,23,23,23,7E,B7,28,03,23,18,F9,23,EB,2A,76,F6
170 DATA 73,23,72,EB,5E,23,56,7A,B3,20,F8,23,22,C2,F6,22
180 DATA C4,F6,22,C6,F6,2A,5C,40,AF,C9,52,45,4E,45,57,00
190 DATA 00,00,00,00
```

Para testar, carregue o programa PROG26.BIN, carregue um programa em BASIC qualquer, entre com o comando **NEW** para apagar o programa em BASIC carregado e, por fim, entre com o comando

CALL RENEW

para recuperar integralmente o programa em BASIC apagado.

O exemplo acima, além de útil, é bastante elucidativo quanto ao método de utilização e implementação do comando **CALL** do BASIC. No próximo capítulo, vamos estudar e criar rotinas para a utilização de um periférico muito em moda: a **MEGARAM.**

BIBLIOGRAFIA RECOMENDADA

PROGRAMAÇÃO AVANÇADA EM MSX - EDITORA ALEPH

Capítulo 5

A MEGARAM

Quando no início de 1988 apareceram as primeiras MEGARAMs, o mercado não tomou consciência do poder deste dispositivo. A sua utilização ficou até hoje limitada aos jogos que a utilizam. Após analisar a MEGARAM, fiquei convencido de que se trata de um dispositivo que pode, simplesmente, aumentar em muito a capacidade de processamento do MSX (tanto na versão 1 como na 2). Vejamos, então, os motivos para tal afirmação.

O QUE É A MEGARAM

A MEGARAM nada mais é que um cartucho com 256 Kbytes de memória RAM. O termo MEGA se refere a 1 Megabit, ou seja, 128 Kbytes. Como se vê, o termo mega está mal empregado, já que a *MEGA*RAM apresenta 256 Kbytes, mas deixemos isso para lá. O que acho de espetacular na MEGARAM é o fato de que ela oferece,num único slot, 256 Kbytes, quando sabemos que, pelo sistema de chaveamento normal, só podemos ter 64 Kbytes por slot ou subslot, no caso de expansão de slots. Fica claro, então, que a MEGARAM possui um chaveamento próprio, pois, caso contrário, não conseguiria acessar os 256 Kbytes disponíveis fazendo uso de apenas um slot ou subslot. Isto é simplesmente maravilhoso! Se possuírmos um expansor de slots com 4 MEGARAMs conectadas a cada um dos 4 subslots, podemos alcançar a impressionante quantidade de 1 Megabyte por slot principal! Incrível, não? Ainda mais incrível se considerarmos que o projeto original do MSX só previa a utilização de, no máximo, 1 Megabyte usando-se todos os 16 subslots, cada qual com 64 Kbytes. Imagine as novas possibilidades que a MEGARAM abre. Com esta capacidade de memória, é possível projetar programas com tal complexidade e riqueza de detalhes antes só vistas em micros de 16 e 32 bits. "Poxa, já fiquei empolgado! Mas, como posso acessar tudo isso?"

FUNCIONAMENTO DA MEGARAM

O funcionamento da MEGARAM é inacreditavelmente simples, se você já domina o gerenciamento dos slots no MSX. Você já sabe que um slot ou subslot normal do MSX está dividido em páginas de 16 Kbytes. A MEGARAM é semelhante: está dividida em páginas de 8 Kbytes. Sendo assim, a MEGARAM apresenta um total de 256 Kbytes/ 8 Kbytes = 32 páginas. Vamos, então, ao o estudo do gerenciamento de todas essas páginas.

O gerenciamento da MEGARAM é feito com o auxílio de uma porta de dados (a porta #8E) e dos endereços que delimitam páginas de 8 Kbytes (#0000, #2000, #4000, #6000, #8000, #A000, etc.). Para indicar à MEGARAM que devemos chavear a página de um determinado endereço, devemos, primeiramente, indicar essa operação com a instrução

OUT(#8E),A

sem nos preocuparmos com o valor do registro A. Logo após, devemos indicar à MEGARAM a página e o segmento de memória que receberá o conteúdo de tal página. Assim sendo, suponha que desejamos colocar o conteúdo da página 0 (a numeração das páginas vai de 0 a 31) no segmento de memória que se inicia no endereço #4000 e termina no endereço #7FFF do slot ou subslot onde está instalada a MEGARAM. As instruções necessárias seriam:

```
OUT    (#8E),A        ;PREPARA A MEGARAM PARA CHAVEAMENTO
XOR    A              ;A=PÁGINA 0
LD     (#4000),A      ;COLOCA A PÁGINA 0 ENTRE OS ENDEREÇOS
                      ;#4000 E #7FFF
```

Sendo assim, fica claro que, no caso acima, as rotinas contidas na página 0 da MEGARAM devem fazer deslocamentos (JUMP, CALL, JR) para os endereços entre #4000 e #7FFF. Portanto, devemos tomar muito cuidado na hora de programar tais rotinas. Se, por exemplo, as instruções acima fossem substituídas pelas instruções

```
OUT    (#8E),A        ;PREPARA A MEGARAM PARA CHAVEAMENTO
XOR    A              ;A=PÁGINA 0
LD     (#8000),A      ;COLOCA A PÁGINA 0 ENTRE OS ENDEREÇOS
                      ;#8000 E #9FFF
```

as rotinas na página 0 deveriam fazer deslocamentos para os endereços entre #8000 e #9FFF. Agora que já aprendemos a chavear, vamos aprender a ler e a escrever na MEGARAM. Uma vez chaveada, a MEGARAM estará pronta para ser lida e quase pronta para ser gravada; digo quase porque ainda falta uma instrução para podermos praticar tal ação. Para assinalar à MEGARAM que desejamos escrever numa determinada página, devemos fazer uso da instrução

INA,(#8E)

Suponha, então, que desejamos alterar o primeiro byte da página 0, prevista para ser usada entre os endereços #4000 e #7FFF. Suponha, ainda, que o byte a ser escrito nessa primeira posição da página 0 esteja contido no registro E e seja igual a #EB (as minhas iniciais em dígitos hexadecimais). O conjunto de instruções a usar seria:

```
LD      E,#EB          ;E=VALOR A SER GRAVADO
OUT     (#8E),A        ;PREPARA A MEGARAM PARA CHAVEAMENTO
XOR     A              ;A=PÁGINA 0
LD      (#4000),A      ;COLOCA A PÁGINA 0 ENTRE OS ENDEREÇOS
                       ;#4000 E #7FFF
IN      A,(#8E)        ;PREPARA A MEGARAM PARA ESCRITA
LD      A,E            ;TRANSFERE PARA A O VALOR A SER ESCRITO
LD      (#4000),A      ;ESCREVE O NOVO VALOR NO PRIMEIRO ENDE-
                       ;REÇO DA PÁGINA 0
```

Agora que você já sabe como gerenciar a MEGARAM, que tal construir uma rotina que verifique se existe ou não uma MEGARAM conectada ao MSX?

IDENTIFICANDO A EXISTÊNCIA DE UMA MEGARAM

O algoritmo de busca a ser usado é o seguinte:

1.Obter e salvar a configuração atual dos slots do MSX.

2.Iniciar a busca da MEGARAM pelos slots principais, começando pelo slot 0.

3.Verificar se o slot principal em questão está expandido ou não.

4.Se o slot principal estiver expandido, iniciar uma busca em cada um dos quatro subslots possíveis.

5.Se o slot principal não estiver expandido, fazer a busca da MEGARAM nesse slot.

6.Repetir os passos de 3 a 5 para todos os quatro slots principais.

7.Terminada a busca, habilitar a configuração original de slots salva no passo **1**.

8.Se a busca tiver sido bem- sucedida, voltar ao BASIC com o slot e o subslot (se for o caso) onde foi encontrada a MEGARAM. Caso contrário, voltar com o valor genérico #FFFF para indicar a não existência de uma MEGARAM conectada ao MSX.

O algoritmo para o teste da existência da MEGARAM é o seguinte:

1.Habilitar a página 1 (#4000 a #7FFF) do slot e do subslot (se for o caso de um slot expandido) que estiverem sendo verificados.

2.Ler e salvar o conteúdo da posição #4000 para evitar um possível dano num programa residente em RAM normal.

3.Preparar a MEGARAM para chaveamento da página 0 dela na página 1 do slot e do subslot (se for o caso) sendo analisados.

4.Proceder ao chaveamento.

5. Preparar a MEGARAM para a operação de escrita na página 0 dela.

6.Gravar o valor #EB no primeiro endereço da página 0 da MEGARAM, ou seja, escrever o valor #EB no endereço #4000.

7.Preparar a MEGARAM para chaveamento da página 0 dela na página 1 do slot e do subslot (se for o caso) sendo analisados.

8.Proceder ao chaveamento.

9. Preparar a MEGARAM para operação de escrita na página 0 dela.

10.Ler o valor contido no endereço #4000 e compará-lo com #EB.

11.Se o valor lido no passo 10 for diferente de #EB, chega-se à conclusão de que no slot e no subslot sendo analisados não existe MEGARAM. Se for este o caso, voltar com a flag de carry resetada (em zero).

12.Se o valor lido no passo 10 for igual a #EB, chega-se à conclusão de que no slot e no subslot sendo analisados existe uma MEGARAM conectada. Se este for o caso, voltar com a flag de carry setada (em um).

As rotinas da ROM do MSX a serem usadas são as seguintes:

Nome da rotina	: RDSLT
Endereço inicial	: #000C
Modo de chamada	: CALL
Parâmetros de entrada	: A=identificação do slot HL=endereço a ser lido
Parâmetros de saída	: A=byte lido
Registros alterados	: AF,BC,DE
Nome da rotina	: WRSLT
Endereço inicial	: #0014
Modo de chamada	: CALL
Parâmetros de entrada	: A=identificação do slot HL=endereço a ser modificado E=byte a ser escrito
Parâmetros de saída	: Nenhum
Registros alterados	: AF,BC,D

Tabela 5.1: Rotinas usadas na verificação da existência de uma MEGARAM conectada ao sistema

Nome da rotina	: ENASLT
Endereço inicial	: #0024
Modo de chamada	: CALL
Parâmetros de entrada	: A=identificação do slot HL=endereço
Parâmetros de saída	: Nenhum
Registros alterados	: AF,BC,DE
Nome da rotina	: RSLREG
Endereço inicial	: #0138
Modo de chamada	: CALL
Parâmetros de entrada	: Nenhum
Parâmetros de saída	: A=conteúdo do registro principal dos slots
Registros alterados	: A
Nome da rotina	: WSLREG
Endereço inicial	: #013B
Modo de chamada	: CALL
Parâmetros de entrada	: A=valor a escrever
Parâmetros de saída	: Nenhum
Registros alterados	: Nenhum

Tabela 5.1: Rotinas usadas na verificação da existência de uma MEGARAM conectada ao sistema(continuação)

Cabe ressaltar que o mapeamento por bits usado na identificação de um slot obedece à seguinte regra:

BITS DO REGISTRO A

7	6	5	4	3	2	1	0
Flag	0	0	0	#SSlot		#SPrin	

onde

Flag é um bit que indica se nesta identificação está incluído ou não um número para o slot secundário (#SSLOT)

#SSLOT é um par de dois bits (bits 2 e 3) que indica o número do slot secundário (entre 0 e 3)

#SPrin é um par de dois bits (bits 0 e 1) que indica o número do slot principal (entre 0 e 3)

Terminada a teoria, vamos para as listagens do programa que detecta a presença da MEGARAM no sistema.

Listagem em assembly Z-80 do código-fonte do programa que detecta a presença da MEGARAM:

```
;programa para detectar a presença da MEGARAM
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG27.BIN=PROG27.GEN
;
;onde PROG27.GEN é o nome do arquivo-texto com esta
;listagem

rdslt           equu    #000c
wrslt           equu    #0014
enaslt          equu    #0024
rslreg          equu    #0138
wslreg          equu    #013b
phydio          equu    #0144
bufset          equu    #f351
valtyp          equu    #f663
argusr          equu    #f7f8
exptbl          equu    #fcc1


        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN


                org     #d100


inicio
                di                              ;desabilita as interrupções
                call    rslreg                  ;lê a configuração atual
                ld      (confatual),a           ;salva em confatual
```

```
            xor     a               ;a=slot 0
            ld      hl,exptbl       ;hl aponta para a tabela
                                    ;de expansões de slots
            ld      b,#04           ;b=número de slots
                                    ;principais
loopbusca
            ld      (slotatual),a   ;Salva o slot atual
            ld      (subsatual),a   ;nestes endereços
            bit     7,(hl)          ;o slot atual está
                                    ;expandido?
            jr      nz,slotsecund   ;Sim, faz a procura
                                    ;nos slots secundários
            call    testamega       ;Não, testa a presença da
                                    ;MEGARAM no slot principal
                                    ;atual
            jr      c,fimbusca      ;Se achou, vai para fimbusca
loopbus_1
            inc     hl              ;Se não, repete a busca no
            ld      a,(slotatual)   ;próximo slot
            inc     a               ;
            djnz    loopbusca       ;


naoachou
            ld      hl,#ffff        ;prepara a sinalização
                                    ;de que não achou a
                                    ;MEGARAM
            jr      fimbus_1        ;e volta ao BASIC


fimbusca
            ld      a,(subsatual)   ;obtém o slot/subslot
                                    ;onde encontrou a
                                    ;MEGARAM
            ld      l,a             ;prepara a volta ao
            ld      h,#00           ;BASIC

fimbus_1    ld      a,#02           ;indica parâmetro
            ld      (valtyp),a      ;inteiro
            ld      (argusr),hl     ;e passa a indicação
                                    ;para o BASIC
            ld      a,(confatual)   ;a=config. atual
            call    wslreg          ;restabelece a
                                    ;config. atual
            ei                      ;habilita as interrupções
            ret                     ;retorna ao BASIC
```

```
;a rotina slotsecun promove uma busca da MEGARAM nos
;quatro possíveis slots secundários de um slot
;principal

slotsecund
            push    bc              ;salva os registros
            push    hl              ;afetados
            ld      e,#00           ;e=1o. slot secundário
            ld      b,#04           ;b=4 slots secundários
slotsec_1
            ld      a,e             ;a=núm do slot secund.
            rla                     ;coloca o número nos
            rla                     ;bits 2 e 3
            and     %00001100       ;obtém somente os bits
                                    ;2 e 3
            ld      e,a             ;salva o resultado em e
            ld      a,(slotatual)   ;obtém o slot principal
            and     %00000011       ;certifica-se de estar
                                    ;entre 0 e 3
            or      e               ;a=ID do slot e do subslot
            set     7,a             ;indica que ID contém
                                    ;identificação de subslot
            ld      (subsatual),a   ;salva neste endereço
            call    testamega       ;testa a presença da
                                    ;MEGARAM
            jr      c,fimslotsec    ;Achou, vai para fimslotsec
            inc     e               ;Se não, repete a busca
            djnz    slotsec_1       ;no próximo subslot

fimslotsec
            pop     hl              ;recupera os registros
            pop     bc              ;salvos
            jp      nc,loopbus_1    ;Se não achou, continua
                                    ;a busca noutro slot
            jr      fimbusca        ;Se achou, volta para
                                    ;o BASIC


;a rotina testamega testa a presença da MEGARAM
;num determinado slot/subslot

testamega
            push    bc              ;salva os registros
            push    de              ;afetados
            push    hl              ;
            call    leslot          ;lê o conteúdo da
```

```
                                        ;pos. #4000 do slot ou
                                        ;subslot em questão
            push    af                  ;salva o valor lido
            out     (#8e),a             ;prepara a MEGARAM para
                                        ;chaveamento
            ld      e,#00               ;chaveia a MEGARAM para
            call    escslot             ;a página 0
            in      a,(#8e)             ;prepara a MEGARAM para
                                        ;escrita
            ld      e,#eb               ;e=valor a escrever
            call    escslot             ;escreve
            out     (#8e),a             ;prepara a MEGARAM para
                                        ;chaveamento
            ld      e,#00               ;chaveia a MEGARAM para
            call    escslot             ;a página 0
            call    leslot              ;lê o valor
            cp      #eb                 ;compara com o valor
                                        ;escrito
            jr      z,fimteste          ;Se achou, vai para fimteste

            pop     af                  ;Se não, recupera o valor
            ld      e,a                 ;original e o repõe
            call    escslot             ;
            xor     a                   ;Zera a flag de carry para
                                        ;indicar que não encontrou
                                        ;a MEGARAM
            jr      fimteste_1          ;

fimteste
            pop     af                  ;Repõe o valor
            ld      e,a                 ;original
            call    escslot             ;
            scf                         ;indica que encontrou a
                                        ;MEGARAM
fimteste_1
            pop     hl                  ;recupera os
            pop     de                  ;registros salvos
            pop     bc                  ;
            ret                         ;e retorna


;a rotina leslot lê o conteúdo do endereço
;#4000 de qualquer slot/subslot

leslot
            ld      hl,#4000            ;hl aponta para o
```

```
                                          ;endereço a ler
                ld      a,(subsatual)     ;obtém o ID do slot
                                          ;ou subslot
                call    rdslt             ;lê o valor
                ret                       ;retorna com o valor
                                          ;lido no registro a


;a rotina escslot escreve o conteúdo do registro
;e no endereço #4000 de qualquer slot/subslot

escslot
                ld      hl,#4000          ;hl aponta para o
                                          ;endereço a ser
                                          ;modificado
                ld      a,(subsatual)     ;obtém o ID do slot
                                          ;ou subslot
                call    wrslt             ;escreve o valor
                                          ;contido no registro
                                          ;e
                ret                       ;retorna


confatual       defb    #00               ;armazena a configuração
                                          ;original

slotatual defb  #00                       ;armazena os registros dos
                                          ;slots principais atuais

subsatual       defb    #00               ;armazena o slot secundário
                                          ;atual

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa que detecta a presença da MEGARAM:

```
10 FOR A%=&HD100 TO &HD1B0
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG27.BIN",&HD100,&HD1B0
100 DATA F3,CD,38,01,32,AD,D1,AF,21,C1,FC,06,04,32,AE,D1
110 DATA 32,AF,D1,CB,7E,20,27,CD,64,D1,38,0C,23,3A,AE,D1
```

```
120 DATA 3C,10,EA,21,FF,FF,18,06,3A,AF,D1,6F,26,00,3E,02
130 DATA 32,63,F6,22,F8,F7,3A,AD,D1,CD,3B,01,FB,C9,C5,E5
140 DATA 1E,00,06,04,7B,17,17,E6,0C,5F,3A,AE,D1,E6,03,B3
150 DATA CB,FF,32,AF,D1,CD,64,D1,38,03,1C,10,E7,E1,C1,D2
160 DATA 1C,D1,18,C4,C5,D5,E5,CD,99,D1,F5,D3,8E,1E,00,CD
170 DATA A3,D1,DB,8E,1E,EB,CD,A3,D1,D3,8E,1E,00,CD,A3,D1
180 DATA CD,99,D1,FE,EB,28,08,F1,5F,CD,A3,D1,AF,18,06,F1
190 DATA 5F,CD,A3,D1,37,E1,D1,C1,C9,21,00,40,3A,AF,D1,CD
200 DATA 0C,00,C9,21,00,40,3A,AF,D1,CD,14,00,C9,00,00,00
210 DATA 00
```

Listagem do programa de teste:

```
100 CLS:KEYOFF
110 BLOAD"PROG27.BIN"
120 DEFUSR=&HD100:REM *** ENDEREÇO DE ENTRADA DA ROTINA ***
130 A%=USR(0):REM *** A%=SLOT/SUBSLOT DA MEGARAM ***
140 IF A%=-1 THEN GOTO 350:REM *** NÃO EXISTE MEGARAM ***
150 A$=RIGHT$("00000000"+BIN$(A%),8)
160 IF MID$(A$,1,1)="0" THEN GOSUB 180 ELSE GOSUB 220
170 END
180 GOSUB 290:REM *** IMPRIME O TÍTULO ***
190 LOCATE 0,11:PRINT "SLOT PRINCIPAL  : ";
200 PRINT VAL("&B"+MID$(A$,7,2))
210 RETURN
220 GOSUB 290:REM *** IMPRIME O TÍTULO ***
230 LOCATE 0,11:PRINT "SLOT PRINCIPAL  : ";
240 PRINT VAL("&B"+MID$(A$,7,2))
250 LOCATE 0,12:PRINT "SLOT SECUNDÁRIO : ";
260 PRINT VAL("&B"+MID$(A$,5,2))
270 RETURN
280 REM *** MENSAGEM PRINCIPAL ***
290 S$="LOCALIZAÇÃO DA MEGARAM"
300 IF PEEK(&HF3B0)>40 THEN CT%=80 ELSE CT%=40
310 LOCATE (CT%-LEN(S$))/2,8
320 PRINT S$
330 REM *** MENSAGEM DE ERRO ***
340 RETURN
350 S$="NÃO EXISTE MEGARAM CONECTADA AO SISTEMA"
360 IF PEEK(&HF3B0)>40 THEN CT%=80 ELSE CT%=40
370 LOCATE (CT%-LEN(S$))/2,8
380 PRINT S$
390 END
```

Ao ser executado, o programa em BASIC acima promoverá a busca através da rotina em linguagem de máquina que se situa a partir do endereço #D100. No retorno, a variável A% indica o sucesso ou não da busca, como se pode ver pelo teste feito na linha 140.

UMA APLICAÇÃO ÚTIL DA MEGARAM

Você deve estar lembrado que, no Capítulo 1, afirmei que um método de se acelerar as operações de I/O é usar buffers. Sendo assim, que tal usar a MEGARAM como um superbuffer de 256 Kbytes? Melhor ainda: Que tal usar esse buffer para uma cópia física de disquetes no padrão MSX-DOS ou MS-DOS? Para chegar ao programa que fará tal maravilha, temos de criar mais duas rotinas: uma que permita a transferência de dados da RAM normal para a MEGARAM e outra que permita a transferência de dados no sentido inverso. A idéia é usar um buffer de 16 Kbytes em RAM para o armazenamento temporário de 32 setores do disquete e, depois, descarregar esse buffer em duas páginas da MEGARAM (não se esqueça que o conceito de páginas na MEGARAM envolve blocos de apenas 8 Kbytes). A implementação destas duas rotinas não apresenta nenhum transtorno adicional em relação à implementação da rotina para o teste da existência da MEGARAM (programa 27). As novas rotinas receberam os seguintes nomes:

RAM_MEGA transfere um bloco da RAM normal, entre os endereços #9000 e #CFFF, para duas páginas da MEGARAM entre os endereços #4000 e #7FFF (uma página de #4000 a #5FFF e outra de #6000 a #7FFF).

MEGA_RAM transfere um bloco de 16 Kbytes de duas páginas da MEGARAM, entre os endereços #4000 e #7FFF, para a RAM normal entre os endereços #9000 e #CFFF.

LEBOOT lê o setor de boot (setor 0) para analisar o tipo de disquete.

LECONJ lê um conjunto de até 32 setores (16 Kb) do disquete-fonte para a memória RAM.

GRCONJ grava um conjunto de até 32 setores (16 Kb) no disquete de cópia.

Não se intimide com as rotinas para a leitura e gravação de setores apresentadas nas listagens abaixo. No próximo capítulo, vamos estudar essas e outras rotinas relacionadas com o sistema de disco do MSX.

O PROGRAMA PARA CÓPIA DE SETORES USANDO A MEGARAM

Este programa se compõem de dois módulos: um em linguagem de máquina e outro em BASIC. Reconheço que a listagem em BASIC não ficou muito clara, por eu não ter colocado

comentários e nem espaços separando os comandos. Esta ausência se deve ao fato de que o programa deveria possuir no máximo 2 Kbytes, tendo em vista o carregamento dos setores a ler/escrever a partir do endereço #9400. Vamos então às listagens.

Listagem em assembly Z-80 do código-fonte do programa para cópia de setores:

```
;programa para cópia de setores
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG28.BIN=PROG28.GEN
;
;onde PROG28.GEN é o nome do arquivo-texto com esta
;listagem

rdslt           equ     #000c
wrslt           equ     #0014
enaslt          equ     #0024
rslreg          equ     #0138
wslreg          equ     #013b
phydio          equ     #0144
bufset          equ     #f351
valtyp          equ     #f663
argusr          equ     #f7f8
exptbl          equ     #fcc1


        defb    #fe                     ;simula em CP/M
        defw    inicio                  ;o cabecalho de
        defw    fim                     ;um arquivo
        defw    inicio                  ;.BIN


                org     #d100


inicio
                di                              ;desabilita as interrupções
                call    rslreg                  ;lê a configuração atual
                ld      (confatual),a           ;salva em confatual
                xor     a                       ;a=slot 0
                ld      hl,exptbl               ;hl aponta para a tabela
                                                ;de expansões de slots
                ld      b,#04                   ;b=número de slots
                                                ;principais
loopbusca
```

```
            ld      (slotatual),a       ;Salva o slot atual
            ld      (subsatual),a       ;nestes endereços
            bit     7,(hl)              ;o slot atual está
                                        ;expandido?
            jr      nz,slotsecund       ;Sim, faz a procura
                                        ;nos slots secundários
            call    testamega           ;Não, testa a presença da
                                        ;MEGARAM no slot principal
                                        ;atual
            jr      c,fimbusca          ;Se achou, vai para fimbusca
loopbus_1
            inc     hl                  ;Se não, repete a busca no
            ld      a,(slotatual)       ;próximo slot
            inc     a                   ;
            djnz    loopbusca           ;


naoachou
            ld      hl,#ffff            ;prepara a sinalização
                                        ;de que não achou a
                                        ;MEGARAM
            jr      fimbus_1            ;e volta ao BASIC


fimbusca
            ld      a,(subsatual)       ;obtém o slot/subslot
                                        ;onde encontrou a
                                        ;MEGARAM
            ld      l,a                 ;prepara a volta ao
            ld      h,#00               ;BASIC

fimbus_1    ld      a,#02               ;indica parâmetro
            ld      (valtyp),a          ;inteiro
            ld      (argusr),hl         ;e passa a indicação
                                        ;para o BASIC
            ld      a,(confatual)       ;a=config. atual
            call    wslreg              ;restabelece a
                                        ;config. atual
            ei                          ;habilita as interrupções
            ret                         ;retorna ao BASIC


;a rotina slotsecun promove uma busca da MEGARAM nos
;quatro possíveis slots secundários de um slot
;principal
```

```
slotsecund
            push    bc              ;salva os registros
            push    hl              ;afetados
            ld      e,#00           ;e=1o. slot secundário
            ld      b,#04           ;b=4 slots secundários
slotsec_1
            ld      a,e             ;a=núm do slot secund.
            rla                     ;coloca o número nos
            rla                     ;bits 2 e 3
            and     %00001100       ;obtém somente os bits
                                    ;2 e 3
            ld      e,a             ;salva o resultado em e
            ld      a,(slotatual)   ;obtém o slot principal
            and     %00000011       ;certifica-se de estar
                                    ;entre 0 e 3
            or      e               ;a=ID do slot e subslot
            set     7,a             ;indica que ID contém
                                    ;identificação de subslot
            ld      (subsatual),a   ;salva neste endereço
            call    testamega       ;testa a presença da
                                    ;MEGARAM
            jr      c,fimslotsec    ;Achou, vai para fimslotsec
            inc     e               ;Se não, repete a busca
            djnz    slotsec_1       ;no próximo subslot

fimslotsec
            pop     hl              ;recupera os registros
            pop     bc              ;salvos
            jp      nc,loopbus_1    ;Se não achou, continua
                                    ;a busca noutro slot
            jr      fimbusca        ;Se achou, volta para
                                    ;o BASIC


;a rotina testamega testa a presença da MEGARAM
;num determinado slot/subslot

testamega
            push    bc              ;salva os registros
            push    de              ;afetados
            push    hl              ;
            call    leslot          ;lê o conteúdo da
                                    ;pos. #4000 do slot ou
                                    ;subslot em questão
            push    af              ;salva o valor lido
            out     (#8e),a         ;prepara a MEGARAM para
```

```
                                        ;chaveamento
                ld      e,#00           ;chaveia a MEGARAM para
                call    escslot         ;a página 0
                in      a,(#8e)         ;prepara a MEGARAM para
                                        ;escrita
                ld      e,#eb           ;e=valor a escrever
                call    escslot         ;escreve
                out     (#8e),a         ;prepara a MEGARAM para
                                        ;chaveamento
                ld      e,#00           ;chaveia a MEGARAM para
                call    escslot         ;a página 0
                call    leslot          ;lê o valor
                cp      #eb             ;compara com o valor
                                        ;escrito
                jr      z,fimteste      ;Se achou, vai para fimteste

                pop     af              ;Se não, recupera o valor
                ld      e,a             ;original e o repõe
                call    escslot         ;
                xor     a               ;Zera a flag de carry para
                                        ;indicar que não encontrou
                                        ;a MEGARAM
                jr      fimteste_1      ;
fimteste
                pop     af              ;Repõe o valor
                ld      e,a             ;original
                call    escslot         ;
                scf                     ;indica que encontrou a
                                        ;MEGARAM
fimteste_1
                pop     hl              ;recupera os
                pop     de              ;registros salvos
                pop     bc              ;
                ret                     ;e retorna


;a rotina leslot lê o conteúdo do endereço
;#4000 de qualquer slot/subslot

leslot
                ld      hl,#4000        ;hl aponta para o
                                        ;endereço a ler
                ld      a,(subsatual)   ;obtém o ID do slot
                                        ;ou subslot
                call    rdslt           ;lê o valor
```

```
            ret                         ;retorna com o valor
                                        ;lido no registro a


;a rotina escslot escreve o conteúdo do registro
;e no endereço #4000 de qualquer slot/subslot

escslot
            ld      hl,#4000            ;hl aponta para o
                                        ;endereço a ser
                                        ;modificado
            ld      a,(subsatual)       ;obtém o ID do slot
                                        ;ou subslot
            call    wrslt               ;escreve o valor
                                        ;contido no registro
                                        ;e
            ret                         ;retorna


;transfere um bloco de 16 Kbytes da RAM para a MEGARAM

ram_mega
            di                          ;desabilita as interrupções
            call    inimega             ;inicializa a MEGARAM
            in      a,(#8e)             ;prepara a MEGARAM
                                        ;para escrita
            ld      hl,#9400            ;hl=end. inicial
            ld      de,#4000            ;de=end. final
            ld      bc,#4000            ;bc=16 Kbytes
            ldir                        ;transfere
            out     (#8e),a             ;
            ld      a,(confatual)       ;a=conf. original
            call    wslreg              ;habilita a config.
                                        ;original dos slots
            ei                          ;habilita as interrupções
            ret                         ;retorna ao BASIC


;transfere um bloco de 16 Kbytes da MEGARAM para a RAM

mega_ram
            di                          ;desabilita as interrupções
            call    inimega             ;inicializa MEGARAM
```

```
            ld      hl,#4000        ;hl=end. inicial
            ld      de,#9400        ;de=end. final
            ld      bc,#4000        ;bc=16 Kbytes
            ldir                    ;transfere
            out     (#8e),a         ;
            ld      a,(confatual)   ;a=conf. original
            call    wslreg          ;habilita a config.
                                    ;original dos slots
            ei                      ;habilita as interrupções
            ret                     ;retorna ao BASIC


;a rotina inmega inicializa a MEGARAM.
;O parâmetro da função USR do BASIC
;deve indicar a página da MEGARAM a ser chaveada

inimega
            call    rslreg          ;lê a configuração
                                    ;atual dos slots
            ld      (confatual),a   ;salva em confatual
            ld      a,(subsatual)   ;obtém o ID da MEGARAM
            ld      hl,#4000        ;habilita a página
            call    enaslt          ;1 do slot da MEGARAM
            out     (#8e),a         ;prepara a MEGARAM
                                    ;para chaveamento
            ld      a,(argusr)      ;obtém a página da
                                    ;MEGARAM
            ld      (#4000),a       ;chaveia
            inc     a               ;chaveia a próxima
                                    ;página no endereço
            ld      (#6000),a       ;#6000
            ret                     ;retorna


;a rotina leboot lê os setores 0 do disquete-fonte e de
;destino, para comparar os tipos e obter o número de
;setores a copiar. O parâmetro da função USR passa
;o número do drive a analisar

leboot
            ld      hl,(bufset)     ;obtém o endereço
                                    ;do buffer
            ld      de,#0000        ;de=núm. do setor
                                    ;a ler
            ld      b,#01           ;b=número de setores
```

```
                                      ;a ler
        ld      c,#f9                 ;c=parâmetro de
                                      ;formatação
        xor     a                     ;zera a flag
                                      ;de carry
        ld      a,(argusr)            ;a=drive
        call    phydio                ;lê o boot
        ld      ix,(bufset)           ;obtém o tipo
        ld      a,(ix+#15)            ;de formatação
        ld      (tipofor),a           ;salva em tipofor
        ret                           ;volta ao BASIC


;a rotina leconj lê um conjunto de até 32 setores (16 Kb)
;do disquete-fonte

leconj
        ld      hl,#9400              ;hl=end. inicial
        ld      de,(argusr)           ;obtém o número
                                      ;do setor inicial
        ld      a,(numset)            ;obtém o número
                                      ;de setores a ler
        ld      b,a                   ;coloca em b
        ld      a,(tipofor)           ;obtém o tipo de
        ld      c,a                   ;formatação
        xor     a                     ;zera a flag de carry
        ld      a,(drive)             ;obtém o drive
        call    phydio                ;lê os setores
        ret                           ;retorna ao BASIC


;a rotina grconj grava um conjunto de até 32 setores
;(32 Kb) no disquete de destino

grconj
        ld      hl,#9400              ;hl=end. inicial
        ld      de,(argusr)           ;obtém o número
                                      ;do setor inicial
        ld      a,(numset)            ;obtém o número
                                      ;de setores a ler
        ld      b,a                   ;coloca em b
        ld      a,(tipofor)           ;obtém o tipo de
        ld      c,a                   ;formatação
        ld      a,(drive)             ;obtém o drive
        scf                           ;seta a flag de carry
        call    phydio                ;lê os setores
```

```
                ret                             ;retorna ao BASIC

drive           defb    #00                     ;drive atual

numset          defb    #00                     ;núm. de setores a
                                                ;ler ou gravar

tipofor         defb    #00                     ;tipo de formatação

endini          defw    #0000                   ;end. inicial

endfim          defw    #0000                   ;end. final

confatual       defb    #00                     ;armazena a configuração
                                                ;original

slotatual       defb    #00                     ;armazena o slot principal
                                                ;atual

subsatual       defb    #00                     ;armazena o slot secundário
                                                ;atual

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa para cópia de setores:

```
10 FOR A%=&HD500 TO &HD651
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG28.BIN",&HD500,&HD651
100 DATA F3,CD,38,01,32,4E,D6,AF,21,C1,FC,06,04,32,4F,D6
110 DATA 32,50,D6,CB,7E,20,27,CD,64,D5,38,0C,23,3A,4F,D6
120 DATA 3C,10,EA,21,FF,FF,18,06,3A,50,D6,6F,26,00,3E,02
130 DATA 32,63,F6,22,F8,F7,3A,4E,D6,CD,3B,01,FB,C9,C5,E5
140 DATA 1E,00,06,04,7B,17,17,E6,0C,5F,3A,4F,D6,E6,03,B3
150 DATA CB,FF,32,50,D6,CD,64,D5,38,03,1C,10,E7,E1,C1,D2
160 DATA 1C,D5,18,C4,C5,D5,E5,CD,99,D5,F5,D3,8E,1E,00,CD
170 DATA A3,D5,DB,8E,1E,EB,CD,A3,D5,D3,8E,1E,00,CD,A3,D5
180 DATA CD,99,D5,FE,EB,28,08,F1,5F,CD,A3,D5,AF,18,06,F1
190 DATA 5F,CD,A3,D5,37,E1,D1,C1,C9,21,00,40,3A,50,D6,CD
200 DATA 0C,00,C9,21,00,40,3A,50,D6,CD,14,00,C9,F3,CD,E1
210 DATA D5,DB,8E,21,00,94,11,00,40,01,00,40,ED,B0,D3,8E
220 DATA 3A,4E,D6,CD,3B,01,FB,C9,F3,CD,E1,D5,21,00,40,11
```

```
230 DATA 00,94,01,00,40,ED,B0,D3,8E,3A,4E,D6,CD,3B,01,FB
240 DATA C9,CD,38,01,32,4E,D6,3A,50,D6,21,00,40,CD,24,00
250 DATA D3,8E,3A,F8,F7,32,00,40,3C,32,00,60,C9,2A,51,F3
260 DATA 11,00,00,06,01,0E,F9,AF,3A,F8,F7,CD,44,01,DD,2A
270 DATA 51,F3,DD,7E,15,32,49,D6,C9,21,00,94,ED,5B,F8,F7
280 DATA 3A,48,D6,47,3A,49,D6,4F,AF,3A,47,D6,CD,44,01,C9
290 DATA 21,00,94,ED,5B,F8,F7,3A,48,D6,47,3A,49,D6,4F,3A
300 DATA 47,D6,37,CD,44,01,C9,00,00,00,00,00,00,00,00,00
310 DATA 00,00
```

Listagem do programa em BASIC para cópia de setores:

```
1000 KEYOFF:WIDTH40:CLEAR 200,&H93FF:DEFINTA-Z
1010 CLS:BLOAD"PROG25.BIN",R:BLOAD"PROG28.BIN"
1020 DEFUSR0=&HD500:DEFUSR1=&HD5AD:DEFUSR2=&HD5C8
1030 DEFUSR3=&HD5FD:DEFUSR4=&HD619:DEFUSR5=&HD630
1040 DR=&HD647:NT=&HD648
1050 GOSUB1780
1060 IFA=1THENENDELSECLS
1070 ED!=PEEK(&HF351)+256*PEEK(&HF352)
1080 DEFFNN(ED!)=(PEEK(ED!+19)+256*PEEK(ED!+20))
1090 A=USR0(0):IFA=-1THENGOTO2090
1100 GOSUB1130
1110 IFDD=DFTHENGOSUB1200ELSEGOSUB1340
1120 GOTO 1050
1130 S1$="DRIVE-FONTE":S2$="DRIVE-CÓPIA"
1140 S3$="ESCOLHA OS DRIVES":X1=(40-LEN(S3$))/2-1:X2=X1+1+
LEN(S3$):Y1=9:Y2=11
1150 WINDOWX1,Y1,X2,Y2,1:LOCATEX1+1,Y1+1:PRINTS3$:LOCATE1,0
:PRINTS1$:LOCATE21,0:PRINTS2$:C=0:D=0
1160 MENU1,0,21,0,11,9,A
1170 IFA=0THENGOSUB1480:DF=A:C=1:IFD=1THENRETURNELSEGOTO
1160
1180 IFA=1THENGOSUB1480:DD=A:D=1:IFC=1THENRETURN
1190 GOTO1160
1200 DA=DF:GOSUB1960:CLS
1210 A=USR3(DF):N1=FNN(ED!)
1220 X1=0:X2=24:Y1=2:Y2=8
1230 WINDOWX1,Y1,X2,Y2,1
1240 LOCATEX1+1,Y1+2:PRINT"DISCO-FONTE : ";CHR$(&H41+DF):
LOCATEX1+1,Y2-2:PRINT"NÚM. DE SETORES:";N1
1250 DA=DD:GOSUB1980
1260 A=USR3(DD):N2=FNN(ED!)
1270 X1=12:X2=36:Y1=10:Y2=16
1280 WINDOWX1,Y1,X2,Y2,1
```

```
1290 LOCATEX1+1,Y1+2:PRINT"DISCO DA CÓPIA : ";CHR$(&H41+DD)
:LOCATEX1+1,Y2-2:PRINT"NÚM. DE SETORES:";N2
1300 GOSUB2080
1310 IFN1<>N2THENGOTO1840
1320 GOSUB1590
1330 RETURN
1340 S1$="COLOQUE OS DISQUETES NOS DRIVES"
1350 GOSUB1900:CLS
1360 A=USR3(DF):N1=FNN(ED!):A=USR3(DD):N2=FNN(ED!)
1370 X1=0:X2=24:Y1=2:Y2=8
1380 WINDOWX1,Y1,X2,Y2,1
1390 LOCATEX1+1,Y1+2:PRINT"DRIVE : ";CHR$(&H41+DF)
1400 LOCATEX1+1,Y2-2:PRINT"NÚM. DE SETORES:";N1
1410 X1=X1+12:X2=X2+12:Y1=Y1+8:Y2=Y2+8
1420 WINDOWX1,Y1,X2,Y2,1
1430 LOCATEX1+1,Y1+2:PRINT"DRIVE : ";CHR$(&H41+DD)
:LOCATEX1+1,Y2-2:PRINT"NÚM. DE SETORES:";N2
1440 GOSUB2080
1450 IFN1<>N2THENGOTO1840
1460 GOSUB1590
1470 RETURN
1480 IFA=0THENX1=1ELSEX1=21
1490 X2=X1+8:Y1=1:Y2=4
1500 LOCATEX1-1,Y1+1:PRINT" ":LOCATEX1-1,Y1+2:PRINT" ":
LOCATEX2+1,Y1+1:PRINT" ":LOCATEX2+1,Y1+2:PRINT" "
1510 REVERSE,,0,1
1520 WINDOWX1,Y1,X2,Y2,1
1530 LOCATEX1+1,Y1+1:PRINT"DRIVE A"
1540 LOCATEX1+1,Y1+2:PRINT"DRIVE B"
1550 MENUX1+1,Y1+1,X1+1,Y1+2,7,1,A
1560 REVERSE,,0,1
1570 LOCATEX1-1,Y1+1+A:PRINT">":LOCATEX2+1,Y1+1+A:PRINT"<"
1580 RETURN
1590 Q1=N1-(N1MOD512):Q2=N1-((N1-Q1)MOD32):Q3=N1
1600 FORX=0TOQ1-1STEP512
1610 DA=DF:PG=0:NS=32:POKEDR,DA:POKENT,NS:IFQ1=0THENQL=Q2-1
ELSEQL=X+511
1620 IFDF=DDTHENGOSUB1960
1630 FORY=XTOQLSTEP32:SI=Y:GOSUB2000:A=USR4(Y):A=USR1(PG):
PG=PG+2:NEXTY
1640 IFQ1=0THENNS=N1-Q2:POKENT,NS:SI=Q2:GOSUB2000:A=USR4
(Q2):A=USR1(PG)
1650 DA=DD:PG=0:NS=32:POKEDR,DA:POKENT,NS:IFDF=DDTHENGOSUB
1980
1660 FORY=XTOQLSTEP32:SI=Y:GOSUB2020:A=USR2(PG):A=USR5(Y):
PG=PG+2:NEXTY
1670 IFQ1=0THENNS=N1-Q2:POKENT,NS:SI=Q2:GOSUB2020:A=USR2
```

```
(PG):A=USR5(Q2):GOTO1770
1680 NEXTX
1690 DA=DF:PG=0:NS=32:POKEDR,DA:POKENT,NS:IFDF=DDTHENGOSUB
1960
1700 FORY=Q1TOQ2-1STEP32:SI=Y:GOSUB2000:A=USR4(Y):A=USR1
(PG):PG=PG+2:NEXTY
1710 IF(Q2=Q3ANDDF=DD)THENGOSUB1980
1720 IFQ2=Q3THENPG=0:DA=DD:POKEDR,DA:FORY=Q1TOQ2-1STEP32:
SI=Y:GOSUB2020:A=USR2(PG):A=USR5(Y):PG=PG+2:NEXTY:GOTO1770
1730 NS=Q3-Q2:POKENT,NS:SI=Q2:GOSUB2000:A=USR4(Q2):A=
USR1(PG)
1740 DA=DD:PG=0:NS=32:POKEDR,DA:POKENT,NS:IFDF=DDTHENGOSUB
1980
1750 FORY=Q1TOQ2-1STEP32:SI=Y:GOSUB2020:A=USR2(PG):A=USR5
(Y):PG=PG+2:NEXTY
1760 NS=Q3-Q2:POKENT,NS:SI=Q2:GOSUB2020:A=USR2(PG):A=USR5
(Q2)
1770 RETURN
1780 CLS:S1$="CÓPIA DE DISQUETES? SIM":S2$="NAO"
1790 X1=(40-LEN(S1$))/2:Y1=11:LOCATEX1,Y1:PRINTS1$
1800 X1=X1+LEN(S1$)-3:LOCATEX1,Y1+1:PRINTS2$
1810 MENUX1,Y1,X1,Y1+1,3,1,A
1820 REVERSE,,0,1
1830 RETURN
1840 S1$="O DISQUETE DE CÓPIA É DIFERENTE":S2$="DO DISQUETE-FONTE"
1850 CLS:BEEP:X1=(40-LEN(S1$))/2:X2=X1+1+LEN(S1$):Y1=10
:Y2=13
1860 WINDOWX1,Y1,X2,Y2,1
1870 LOCATEX1+1,Y1+1:PRINTS1$:LOCATE(40-LEN(S2$))/2,
Y1+2:PRINTS2$
1880 GOSUB2080:GOSUB1780
1890 IFA=1THENCLS:END:ELSECLS:RETURN1070
1900 S2$=" PRESSIONE QUALQUER TECLA"
1910 X1=(40-LEN(S1$))/2-1:X2=X1+1+LEN(S1$):Y1=20:Y2=23
1920 GOSUB2070
1930 WINDOWX1,Y1,X2,Y2,1
1940 LOCATEX1+1,Y1+1:PRINTS1$:LOCATE(40-LEN(S2$))/2,Y2-1:PRINTS2$
1950 IFINKEY$=""THEN1950ELSEGOSUB2070:RETURN
1960 S1$="COLOQUE O DISQUETE-FONTE EM "+CHR$(&H41+DA)+" "
1970 GOSUB1900:RETURN
1980 S1$="COLOQUE O DISQUETE DE CÓPIA EM "+CHR$(&H41+DA)+" "
1990 GOSUB1900:RETURN
2000 S1$="LENDO DO SETOR"+STR$(SI)+" AO"+STR$(SI+NS-1)
2010 GOSUB2040:RETURN
2020 S1$="GRAVANDO DO SETOR"+STR$(SI)+" AO"+STR$(SI+NS-1)
2030 GOSUB2040:RETURN
2040 CLS:X1=(40-LEN(S1$))/2-1:X2=X1+1+LEN(S1$):Y1=10:Y2=12
```

```
2050 WINDOWX1,Y1,X2,Y2,0
2060 LOCATEX1+1,Y1+1:PRINTS1$:RETURN
2070 FORA=Y1TOY2:LOCATE0,A:PRINTSTRING$(39," ");:NEXTA:RETURN
2080 S1$="PRESSIONE QUALQUER TECLA":S2$="PARA CONTINUAR":
GOSUB1910:RETURN
2090 S1$="SISTEMA SEM MEGARAM CONECTADA":S2$="IMPOSSIVEL CONTI-
NUAR":GOSUB1910:CLS:END
```

Apesar de confusa visualmente, a listagem em BASIC acima não é difícil de ser entendida. A maior parte das sub-rotinas é dedicada a exibir mensagens ou menus, sendo que apenas três lidam com o acesso ao disco: uma sub-rotina para comparar os disquetes na cópia usando apenas um drive, outra para fazer a mesma comparação usando dois drives e, por último, uma rotina dedicada única e exclusivamente à cópia.

Ao executar o programa acima, você vai perceber uma disparidade entre a velocidade do processo de leitura, que é razoavelmente rápido, e a velocidade do processo de gravação, que é muito lento. Tal disparidade será corrigida no Capítulo 7, quando estudaremos o acesso direto dos drives.

Com o programa acima, terminamos o estudo da MEGARAM num exemplo prático. As rotinas apresentadas para o acesso da MEGARAM são extremamente flexíveis, sendo por isso mesmo facilmente adaptadas a novas condições. No próximo capítulo, vamos estudar o ambiente do MSX-DOS e as rotinas que ele oferece.

BIBLIOGRAFIA RECOMENDADA

INTRODUÇÃO À LINGUAGEM DE MÁQUINA PARA MSX - Eduardo Alberto Barbosa - Rio de Janeiro - CIÊNCIA MODERNA COMPUTAÇÃO LTDA.

APROFUNDANDO-SE NO MSX - EDITORA ALEPH

PROGRAMAÇÃO AVANÇADA EM MSX - EDITORA ALEPH

Capítulo 6

O SISTEMA DOS

Apesar de não ser uma interface das mais amigáveis, pois necessita que o usuário conheça os comandos de antemão, o sistema **DOS** (do inglês **D**isk **O**perating **S**ystem=Sistema Operacional de Disco) é bastante poderoso e versátil. "Mas, se é tão poderoso e versátil, por que não é amigável?" Ocorre que o sistema operacional para o MSX (o MSX-DOS) está baseado no sistema CP/M, criado nos fins da década de 70. Ora, os microcomputadores daquela época não possuíam nem um terço da capacidade gráfica, ou mesmo de processamento, dos micros atuais. A versatilidade e o poder do CP/M não podem ser contestados, pois trata-se de um sistema que quebrou as barreiras entre os diversos padrões de microcomputadores da época (TRS-80, Apple, etc). O que se pode contestar é a sua adaptação a microcomputadores que oferecem telas coloridas em alta resolução, maior capacidade de armazenamento em disquetes e maior velocidade de processamento. Não podemos nos esquecer que o padrão de armazenamento nos fins da década de 70 eram os disquetes de 5 1/4" com 180 Kbytes formatados. Ora, hoje já temos disquetes armazenando 1,44 MBytes. "Mas, então, por que o sistema operacional escolhido para o MSX foi o CP/M?" Acredito que tal escolha se deve a um motivo básico: **custo**. Por que inventar um novo sistema operacional para um micro de 8 bits (não podemos deixar de levar em conta que, quando o MSX foi lançado, os micros de 16 bits já estavam se impondo com um padrão) se já existia um pronto usando o mesmo processador, amplamente testado e com uma farta biblioteca de programas? Acho, sinceramente, que foram esses os motivos que pesaram na escolha de um sistema operacional para o MSX. "Então, quer dizer que nunca poderei ter uma interface amigável do tipo Windows no meu MSX?" Não exatamente. Vejamos o caso do MS-DOS versão 4.01, que nada mais é do que um conjunto de programas que, sendo executados sob o DOS, implementam uma interface gráfica bastante amigável. No MSX, pode-se repetir o mesmo efeito sem grandes malabarismos, a não ser a possível exigência de um acessório tipo MEGARAM, para suprir as necessidades de memória de tal tipo de programa. Mas, para implementá-lo, temos de conhecer o que oferece o MSX-DOS.

O MSX-DOS

Você pode ficar surpreso, mas a parte principal do MSX-DOS não está contida no chamado disquete do sistema, e sim numa memória ROM (somente para leitura) contida no cartucho da interface do drive. É nessa memória que se encontram as rotinas para o acesso ao drive, entre elas a rotina para leitura/gravação de setores, a rotina para formatação de disquetes e diversas rotinas para o gerenciamento dos arquivos. "Mas, então, o que fazem os programas

do disquete de sistema?" Simples: O disquete de sistema do MSX é composto basicamente por dois arquivos:

MSXDOS.SYS que inicializa o ambiente do DOS e carrega o interpretador de comandos.

COMMAND.COM que é o responsável pela interpretação e execução de comandos como DIR, COPY, etc.

Para funcionar, o arquivo COMMAND.COM faz chamadas constantes às rotinas contidas na ROM da interface do drive. Feitos os esclarecimentos, vamos à apresentação das funções disponíveis no DOS. Antes, porém, cabe ressaltar que tal conjunto de funções recebe o nome genérico de **BDOS**.

AS FUNÇÕES DO BDOS

Estas funções estão disponíveis tanto no MSX-DOS como no BASIC de Disco, variando somente o endereço de chamada.

	COMANDO DE CHAMADA DO BDOS
MSX-DOS	CALL #0005
BASIC DE DISCO	CALL #F37D

Todas as funções abaixo foram implementadas visando uma compatibilidade quase que total com o ambiente CP/M. Digo quase total porque existem algumas pequenas diferenças a nível de endereços de memória, tamanhos de buffers, etc., mas que no conjunto não chegam a comprometer a pseudocompatibilidade. O fato é que conseguimos executar no MSX-DOS os grandes sucessos do CP/M, como Supercalc 2, Dbase II, Turbo Pascal, etc.

Para se ativar uma das funções abaixo, deve-se adotar o seguinte procedimento:

1.Carregar o registro C com o número da função a ser executada.

2.Carregar os registros DE e HL (se for o caso) com os valores apropriados.

3.Fazer uma chamada para o endereço inicial do BDOS (#0005 no MSX-DOS e #F37D no BASIC de Disco)

Função #00 : Esta função provoca o reinício do sistema. No caso da saída de um programa demasiadamente grande e que, por isso, alterou a memória de forma imprevisível, torna-se necessária a chamada desta função para o carregamento do arquivo MSXDOS.SYS ou equivalente. De forma geral, esta função limpa todos os buffers e retorna para o modo de comando do DOS (A>). No BASIC de Disco, esta função simplesmente reinicializa todo o sistema com um efeito igual ao de desligar e ligar o computador.

Função #01 : Esta função faz a leitura de uma tecla com eco na tela. O termo eco está relacionado ao fato de que o caractere correspondente à tecla lida é também impresso na tela. O código ASCII do caractere da tecla lida retorna no registro A.

Função #02 : Esta função imprime na tela o caractere cujo código ASCII se encontra no registro E. Você também pode usar esta função para impressão de caracteres de controle, já que o terminal emulado pelo MSX é o VT-52. Os códigos de controle deverão ser precedidos do código escape (#1B=27), de modo que você deverá, neste caso, fazer duas chamadas a esta função: uma com o código escape e outra com o código do caractere de controle. Os códigos de controle são:

ASCII	HEXA	AÇÃO
ESC,J	1B,6A	Limpa a tela
ESC,E	1B,45	Limpa a tela
ESC,K	1B,4B	Limpa do cursor ao fim da linha
ESC,J	1B,4A	Limpa do cursor ao fim da tela
ESC,l	1B,6C	Limpa toda a linha onde se encontra o cursor
ESC,L	1B,4C	Insere uma linha em branco
ESC,M	1B,4D	Apaga a linha onde se encontra o cursor
ESC,Y,SP+L,SP+C	1B,59,20+L,20+C	Posiciona o cursor nas coordenadas L,C, onde L=linha e C= coluna da tela
ESC,A	1B,41	Move o cursor uma linha para cima
ESC,B	1B,42	Move o cursor uma linha para baixo
ESC,C	1B,43	Move o cursor uma coluna para a direita
ESC,D	1B,44	Move o cursor uma coluna para a esquerda

ESC,H	1B,48	Coloca o cursor nas coordenadas (0,0)
ESC,x,4	1B,78,34	Cursor cheio
ESC,y,4	1B,79,34	Cursor sublinhado
ESC,x,5	1B,78,35	Acende o cursor
ESC,y,5	1B,79,35	Apaga o cursor

Função #03 : Esta função espera a chegada de um caractere pela porta serial (modem, por exemplo), cujo código ASCII será colocado no registro A. Durante a execução desta rotina, é possível o uso da combinação CONTROL-S no teclado para suspender a leitura da porta serial. Infelizmente, esta função não funciona nos modems que não seguem o padrão MSX (até o fechamento deste livro, nenhum modem para o MSX seguia o padrão).

Função #04 : Esta função envia para a porta serial o caractere cujo código ASCII se encontra no registro E. Permanecem válidas as duas observações feitas na função anterior.

Função #05 : Esta função envia para a impressora (porta paralela) o caractere cujo código ASCII se encontra no registro E. Da mesma forma que nas duas rotinas anteriores, é feita uma verificação do teclado para detectar um possível CONTROL-S que suspenda o envio do caractere para a impressora.

Função #06 : Esta função faz uma leitura direta do teclado com funcionamento semelhante ao da função INKEY$ do BASIC. Se na chamada a esta função o registro E armazenar o valor #FF, na volta o registro A conterá o código ASCII da tecla pressionada. Mas,se na chamada o registro E armazenar qualquer valor diferente de #FF, esta função imprimirá primeiramente o caractere cujo código ASCII está na tela, procedendo em seguida à leitura do teclado, com o código ASCII da tecla pressionada retornando em A. Se o registro A voltar com o valor #00, assume-se que não foi pressionada nenhuma tecla.

Função #07 : Esta função faz uma leitura direta do teclado com espera. O seu funcionamento é muito semelhante ao da função INPUT$(1) do BASIC. O código ASCII da tecla pressionada retorna no registro A.

Função #08 : Esta função é idêntica à função #01, diferindo apenas no eco para a tela, pois o caractere lido não é enviado para ela.

Função #09 : Esta função imprime uma string na tela. Na chamada, o par de registro DE deverá apontar para o endereço de memória que contém o primeiro caractere da string a ser impressa. A string deverá terminar com o caractere $ (#24), que não será impresso.

Função #0A : Esta função lê toda uma linha, funcionando de modo semelhante à função LINE INPUT do BASIC. Ela apresenta a vantagem de podermos definir o comprimento da linha a ser lida. Assim sendo, o par DE na chamada deve apontar para um buffer com a seguinte estrutura:

Primeiro byte : Número de caracteres a ler
Segundo byte : Número de caracteres lidos
Terceiro byte : Primeiro caractere lido
Quarto byte : Segundo caractere lido
...

Fica claro que os caracteres lidos serão colocados a partir da terceira posição do buffer. A leitura termina assim que seja pressionada a tecla Enter/Return. Note que devemos fornecer na primeira posição do buffer o número máximo de caracteres a ler. Se o número de caracteres lidos ultrapassar esse limite, o sistema emite um beep para cada caractere extra, sendo que todos os extras serão ignorados. Note que você poderá usar a tecla de BackSpace (BS) para editar a entrada. No retorno, a segunda posição do buffer indicará o número de caracteres lidos.

Função #0B : Esta função faz uma verificação direta do teclado nos moldes da função INKEY$ do BASIC. Se o registro A apresentar o valor #FF no retorno, alguma tecla foi pressionada; já se apresentar o valor #00, nenhuma tecla foi pressionada. Esta função detecta as teclas de controle, como CONTROI-P e CONTROL-C, tomando as decisões apropriadas.

Função #0C : Esta função retorna com o número da versão do DOS instalado. No caso do MSX-DOS, o registro HL retorna com #0022, indicando uma compatibilidade com o sistema CP/M versão 2.2.

Função #0D : Esta função promove uma atualização dos dados do disquete (tipo, número de arquivos, etc.) contidos nos buffers do MSX-DOS. Tenha o cuidado de usar esta função somente na troca de disquetes, pois todos os buffers (incluindo os FCBs) serão apagados.

Função #0E : Esta função seleciona um drive como default. O registro E deverá armazenar o drive (1=A, 2=B, etc.) que se tornará o default.

Função #0F : Esta função abre um arquivo. O par DE deve apontar para o FCB do arquivo que se deseja abrir. Note que esta operação é absolutamente necessária para se iniciar uma leitura. Se o registro A apresentar o valor #00 no retorno, o arquivo existe; já se apresentar o valor #FF, o arquivo não foi encontrado. Por esta razão, essa função é muito utilizada para se descobrir se um determinado arquivo existe ou não. Só um lembrete: o par DE deve apontar para o FCB de um arquivo não aberto.

Função #10 : Esta função deve ser executada após a etapa de gravação de registros num arquivo. Na entrada, o par DE deve apontar para o FCB do arquivo recém-gravado. É absolutamente indispensável chamar esta função após a gravação de novos registros em qualquer arquivo, caso contrário, a entrada do arquivo correspondente no diretório não será atualizada, o que, por sua vez, conduzirá à perda de todos os dados do arquivo em questão.

Função #11 : Esta função procura o primeiro arquivo no diretório cujo nome combine com a descrição contida no FCB apontado pelo par DE. A descrição contida no FCB pode fazer uso do caractere coringa "?". Se o registro A apresentar o valor #00 no retorno, foi encontrado um arquivo que corresponde à descrição feita no FCB, caso contrário, o registro A retorna com o valor #FF. O FCB do arquivo encontrado é colocado a partir do primeiro byte do DMA.

Função #12 : Esta função procura o próximo arquivo no diretório cujo nome combine com a descrição contida no FCB apontado pelo par DE. Permanecem válidas todas as observações feitas na função #11.

Função #13 : Esta função deleta o arquivo cujo FCB é apontado pelo par DE. Se o registro A apresentar o valor #00 no retorno, o arquivo foi deletado com sucesso, caso contrário, o registro A retorna com o valor #FF. Observe que o FCB pode conter o caractere coringa "?", sendo que, neste caso, serão apagados todos os arquivos cujos nomes se correspondam com a especificação contida no FCB.

Função #14 : Esta função lê um bloco de 128 bytes de um arquivo seqüencial cujo FCB é apontado pelo par DE. O bloco de 128 bytes é colocado no DMA. Se o registro A apresentar o valor #00 no retorno, o bloco foi lido com sucesso; se apresentar o valor #01, foi lido o último bloco de 128 bytes do arquivo e, por último, se apresentar o valor #02, houve algum tipo de erro de leitura.

Função #15 : Esta função grava um bloco de 128 bytes num arquivo seqüencial cujo FCB é apontado pelo par DE. Os 128 bytes a serem gravados devem estar no DMA. Se o registro A apresentar o valor #00 no retorno, a gravação foi feita com sucesso; já se apresentar o valor #01, há falta de espaço livre no disco. Qualquer outro valor indica uma situação de erro de gravação.

Função #16 : Esta função cria um arquivo no disco. Na entrada, o par DE deve apontar para o FCB do arquivo a ser criado. Se o registro A apresentar o valor #00 no retorno, o arquivo foi criado com sucesso; já se apresentar o valor #FF, não existe mais espaço no diretório para a criação de um novo arquivo.

Função #17 : Esta função renomeia um ou mais arquivos. Na entrada, o par DE deve apontar para o FCB do arquivo ou arquivos (no caso do FCB conter o caractere coringa "?") a serem renomeados. Na primeira posição do FCB, devemos colocar o drive (1=A, 2=B, etc.) do arquivo a ser renomeado, seguido do seu nome entre as posições 2 e 12. A partir da décima oitava (FCB+#11) posição, devemos colocar o novo nome que o arquivo receberá. Se o registro A apresentar o valor #FF no retorno, o arquivo não foi encontrado; já se apresentar o valor #00, o arquivo foi encontrado e renomeado com sucesso.

Função #18 : Esta função retorna com os 16 bits do par HL setados de acordo com os drives presentes no sistema. Assim sendo, o bit 0 de L representa o drive A, o bit 1 o drive B e assim por diante, até um máximo de 16 drives.

Função #19 : Esta função retorna no registro A com o número do drive default atual (0=A, 1=B, etc.).

Função #1A : Esta função define o endereço inicial do DMA. Na chamada, o par DE deve conter o endereço de memória a ser ocupado pelo DMA. No reset de disco (função #0D) e na inicialização do sistema, o DMA é ajustado automaticamente para o endereço #0080.

Função #1B : Esta função retorna com algumas informações a respeito do disquete no drive a ser analisado. Na entrada, o registro E deve conter o número do drive (1=A, 2=B, etc.). Se o registro A apresentar o valor #FF no retorno, a especificação feita para o drive não foi válida,

caso contrário, o registro A retorna com o número de setores por aglomerado, o par BC com o tamanho do setor em bytes (512 bytes), o par DE com o número total de aglomerados no disquete, o par HL com o número de aglomerados livres, o registro IY aponta para uma cópia da FAT do disquete na RAM e, por último, o registro IX aponta para uma área (BPB) na RAM que contém as informações necessárias sobre o disquete no drive analisado.

Função #1C : Esta função não está implementada no MSX-DOS. No CP/M, ela ativa a proteção contra escrita de um determinado disquete.

Função #1D : Esta função não está implementada no MSX-DOS. No CP/M, ela retorna no par HL com os drives protegidos contra gravação.

Função #1E : Esta função não está implementada no MSX-DOS. No CP/M, ela retorna com os atributos (somente para leitura, etc.) do arquivo cujo FCB é apontado pelo par DE.

Função #1F : Esta função não está implementada no MSX-DOS. No CP/M, ela retorna no par HL o endereço de memória onde estão os parâmetros do disquete no drive default.

Função #20 : Esta função não está implementada no MSX-DOS. No CP/M, ela retorna com o código do usuário.

Função #21 : Esta função ativa a leitura randômica de um registro do arquivo cujo FCB é apontado pelo par DE. Ao ativar esta função pela primeira vez, devemos colocar nas posições 14 e 15 (#0E e #0F) do FCB o tamanho do registro a ser lido, e nas posições de 32 a 36 (#20 a #24) o número do registro, começando pelo valor zero. Assim sendo, se quiséssemos ler o registro 0 (primeiro registro) de um arquivo onde os registros apresentam o tamanho de um byte, teríamos de colocar na posição 14 do FCB o valor #00, na posição 15 o valor #01 e nas posições de 32 a 36 o valor #00. Se o registro A apresentar o valor #00 no retorno, a leitura foi bem-sucedida; se o valor for #01, foi lido o último registro do arquivo e, por último, se o valor for #02, houve problemas na leitura.

Função #22 : Esta função ativa a gravação randômica de um registro para o arquivo cujo FCB é apontado pelo par DE. Os parâmetros de entrada desta rotina são os mesmos da rotina #21. Se o registro A apresentar o valor #01 no retorno, o disco não apresenta mais espaço livre para armazenamento; se o valor for #00, a operação foi completada com sucesso e, por último, no caso de qualquer outro valor, houve algum tipo de erro de gravação.

Função #23 : Esta função retorna com o tamanho do arquivo nas posições de 32 a 36 do FCB apontado pelo par DE. Se o registro A apresentar o valor #00 no retorno, a operação foi completada com sucesso; já se apresentar o valor #FF, o arquivo não foi encontrado.

Função #24 : Esta função lê o próximo registro randômico do arquivo cujo FCB é apontado pelo par DE. A leitura é feita após o incremento do número do registro no FCB.

Função #25 : Esta função não está implementada no MSX-DOS. No CP/M, ela serve para reiniciar o drive após uma parada prolongada ou uma troca de disquetes.

Função #26 :Esta função faz a gravação de vários registros randômicos de uma só vez. Na entrada, o par DE deve, como sempre, apontar para o FCB do arquivo. Observe que, no caso de registros randômicos, existe um campo do FCB que indica o tamanho de cada registro e outro que indica o número do registro atual. Baseado no tamanho de cada registro e no número do registro atual contidos no FCB, o sistema grava um determinado número de registros passados pelo par HL. Se o registro A apresentar o valor #01 no retorno, não existe espaço suficiente no disquete para gravação dos registros; já se apresentar o valor #00, a gravação foi bem-sucedida. Ao retornar da função, o par HL apresenta o número de registros efetivamente gravados. No caso de falta de espaço no disquete, o arquivo fica setado com o tamanho dado pelo FCB dele.

Função #27 : Esta função faz a leitura de vários registros randômicos de uma só vez. Na entrada, o par DE aponta para o FCB do arquivo, e o par HL contém o número de registros a ler. Para esta função valem os comentários feitos na função #26 sobre os campos do FCB relacionados aos arquivos randômicos. Se o registro A apresentar o valor #01 no retorno, foi encontrado o fim do arquivo (EOF); já se apresentar o valor #00, foram lidos com sucesso todos os registros indicados por HL. Ao retornar desta função, o par HL apresenta o número de registros lidos.

Função #28 : Até o fechamento deste livro, não consegui encontrar uma utilidade prática para esta função que, ao que parece, funciona de modo semelhante à função #22, embora preencha os bytes restantes do DMA (considerando um tamanho de 128 bytes para o DMA) com zeros.

Função #29 : Esta função não está implementada no MSX-DOS.

Função #2A : Esta função retorna com a data atual. O par HL contém o ano; o registro D o mês (1=Janeiro, 2=Fevereiro, etc.); o registro E o dia; e o registro A o dia da semana (0=Domingo, 1=Segunda-feira, etc.).

Função #2B : Esta função modifica a data atual. Os pares DE e HL deverão conter a informação necessária, conforme descrito na função #2A.

Função #2C : Esta função retorna com a hora atual. O registro H contém as horas; o L os minutos; o D os segundos; e o E os centésimos de segundo.

Função #2D : Esta função acerta a hora atual. Os pares DE e HL deverão conter a informação necessária, conforme descrito na função #2C.

Função #2E : Esta função ativa e desativa a verificação de escrita. Infelizmente, varia de sistema para sistema, de modo que aconselho você a evitá-la.

Função #2F : Esta função faz uma leitura direta de setores. Na entrada, o par DE deverá apresentar o número do primeiro setor a ler; o registro H o número de setores a ler;e o registro L o drive (0=A, 1=B, etc.) que contém o disquete a ser lido. Os setores serão colocados seqüencialmente a partir do DMA. No retorno, a flag de carry indica se houve ou não um erro de leitura (C=erro, NC=Sem erro).

Função #30 : Esta função realiza uma gravação dos setores armazenados na memória a partir do DMA. Os parâmetros de entrada e de saída são exatamente os mesmos da função #2F.

Apresentadas as funções, vamos passar à análise da estrutura do FCB no MSX-DOS (e, por tabela, no BASIC de Disco) que, como vimos, é uma peça fundamental na manipulação dos arquivos.

A ESTRUTURA DO FCB

O FCB vem da abreviação de **File Control Block**, que em inglês quer dizer Bloco de Controle de Arquivo. O nome é bastante apropriado. A função realizada por este pequeno buffer, que ocupa 36 bytes da memória RAM, é a de controlar permanentemente as informações vitais de um arquivo: o tamanho e o tipo, a data e a hora de criação, etc. Vejamos, então, a sua estrutura:

Byte	Função
0	Número do drive: 0=default, 1=drive A, 2=drive B, etc.
1 a 8	Nome do arquivo
9 a 11	Extensão do arquivo
12	Campo de extensão
13	Reservado
14	Valor LSB do tamanho do registro randômico
15	Valor MSB do tamanho do registro randômico
16 a 19	Tamanho do arquivo em bytes
20 a 21	Data da criação ou modificação do arquivo: Bits F a 9=ano (0=1980, 119=2099) Bits 8 a 5=mês (1 a 12) Bits 4 a 0=dia (1 a 31)
22 a 23	Horário da criação ou modificação do arquivo: Bits F a B=horas (0 a 23) Bits A a 5=minutos (0 a 59) Bits 4 a 0=segundos (0-29)
24 a 31	Reservados
32	Número do registro relativo aos 128 bytes do bloco de um arquivo seqüencial
33 a 36	Número do registro randômico

Tabela 6.1: A estrutura do FCB.

Para completar o conjunto de informações necessárias à construção de programas usando o sistema de disco, precisamos ter acesso a algumas informações contidas no setor de boot (o setor zero) do disquete.

O SETOR DE BOOT NO MSX-DOS

O setor de boot está organizado em duas áreas: uma área de dados, que se estende da posição #00 à posição #1D, e uma área com a rotina de partida (boot), que se estende da posição #1E em diante. Vamos começar pela área de dados:

Byte	Função
#00	Instrução de partida no DOS (utilizada no boot a quente)
#03 a #0A	Nome do fabricante em ASCII. Esses bytes podem ser alterados pelo usuário
#0B	Valor LSB do número de bytes por setor
#0C	Valor MSB do número de bytes por setor
#0D	Número de setores por aglomerado
#0E	Valor LSB do número de setores reservados
#0F	Valor MSB do número de setores reservados
#10	Número de FATs usadas no disquete
#11	Valor LSB do número máximo de entradas no diretório
#12	Valor MSB do número máximo de entradas no diretório
#13	Valor LSB do número total de setores
#14	Valor MSB do número total de setores
#15	Identificação do tipo de formatação do disquete
#16	Valor LSB do número de setores por FAT
#17	Valor MSB do número de setores por FAT
#18	Valor LSB do número de setores por trilha
#19	Valor MSB do número de setores por trilha
#1A	Valor LSB do número de faces do disco
#1B	Valor MSB do número de faces do disco
#1C	Valor LSB do número de setores ocultos
#1D	Valor MSB do número de setores ocultos

Tabela 6.2: Os dados no setor de boot

Se você estranhar os termos utilizados na Tabela acima, sugiro a leitura do livro mais completo no assunto: ***Sistemas Operacionais do MSX & Suas Ferramentas***, de Sérgio Guy Pinheiro Elias e Paulo Roberto Pinheiro Elias. Como você pode perceber, existem muitas informações contidas no setor de boot de um disquete. Infelizmente, o sistema MSX-DOS ignora por completo todas elas. O único byte de informação utilizado pelo MSX-DOS é o primeiro byte da FAT, que é uma repetição do valor contido na posição #15 do setor de boot. A presença de tais dados na rotina de boot foi mantida para permitir uma compatibilidade a nível de discos com o MS-DOS, utilizado na família IBM-PC. É realmente uma pena que o sistema MSX-DOS ignore tais dados, pois isso abriria a possibilidade de se formatar um disquete com 42 trilhas e 10 setores por trilha, por exemplo. Embora o MSx-DOS não utilize tais dados, nada impede que nós façamos um uso inteligente deles. As informações existem e são coerentes, logo, por que não utilizá-las? Suponha, por exemplo, que você deseje saber o setor inicial e o setor final do diretório de um determinado disquete. O que fazer?

Simples: basta obter, em primeiro lugar, o número de setores ocultos e somá-lo ao resultado da multiplicação do número de FATs pelo número de setores por FAT, para obter assim o setor inicial do diretório. Vejamos o caso de um disquete de 5 1/4" de face dupla (360 Kb):

Número de setores ocultos = 1
Número de setores por FAT = 2
Número de FATS = 2
Total de setores = 5

Como o setor inicial recebe o número 0, o início do diretório se situa no setor (0+5)=5. "Já entendi! Mas, como vamos achar o setor final do diretório?" Basta obter o número total de entradas no diretório, multiplicar esse número por 32, que é o número de bytes gasto por cada arquivo no diretório, e depois dividir o resultado por 512, que, por sua vez, é o número de bytes por setor num disquete padrão MS-DOS (se você quiser ser mais rigoroso, também poderá obter este último valor das posições #0B e #0C do setor de boot). Vejamos esses cálculos no caso de um disquete do mesmo tipo usado no exemplo anterior:

Número de entradas = 112
Bytes por arquivo no dir. = 32
Bytes por setor = 512

Número de setores gastos pelo diretório = (112*32)/512=7

Logo, se o diretório se inicia no setor 5 e gasta 7 setores, então termina no setor 11. Conseqüentemente, os arquivos começam a ser armazenados a partir do setor 12. "Mas, qual é a utilidade de se saber o setor inicial e o setor final do diretório?" A utilidade aparecerá num programa para ordenação do diretório de qualquer disquete. Agora que já esgotamos a área de dados do setor de boot, vamos ao estudo da rotina de partida.

A rotina de partida se inicia na posição #1E do setor de boot e realiza as seguintes tarefas:

1.Tenta localizar o arquivo MSXDOS.SYS ou equivalente (SOLXDOS.SYS, etc.). Para tanto, conta com um FCB (ou dois) desse arquivo.

2.Se conseguir localizar o arquivo de sistema no passo 1, carrega esse arquivo a partir da posição #0100 da memória RAM. Ao terminar o carregamento, dá um salto até a posição #0100 para executar o arquivo recém-carregado.

3.Se não conseguir localizar o arquivo de sistema, faz um desvio até uma rotina na ROM da interface do drive para inicializar o BASIC de Disco.

Como você pode perceber, a rotina de partida é extremamente simples. "Mas, como é que o sistema chega a esta etapa?" Na verdade, quando o computador é ligado o MSX inicia uma busca de cartuchos de expansão. Nessa busca, acaba encontrando um cartucho correspondente à interface do drive. Ao encontrar tal cartucho, o MSX é levado a executar o programa contido na ROM dessa interface. Esse programa se inicia no endereço #4000. Após inicializar todas as variáveis a serem usadas pelo sistema de disco, o programa contido na ROM da interface habilita todo o slot com 64 Kbytes de RAM e executa uma pequena rotina que ele mesmo transferiu para a página 3 da memória RAM (entre os endereços #C000 e #FFFF). Essa rotina, na página 3 da memória RAM, tem como função ler o setor 0 do disquete colocado no drive A. Para tanto, promove a leitura do setor 0 e desloca a rotina de partida (contida a partir da posição #1E do setor 0) para o endereço #C000 da memória RAM. É a partir deste endereço que a rotina de boot é executada. Após esta etapa, temos a realização dos passos 1, 2 e 3, descritos acima.

Já estamos ficando mestres em sistema de disco, não? Para terminar o nosso *mestrado*, só está faltando estudar a estrutura do diretório.

A ESTRUTURA DO DIRETÓRIO

O diretório é, por assim dizer, um referencial dos arquivos contidos no disquete. Esta área é extremamente importante para a manutenção da ordem dos arquivos armazenados nos discos. Vamos então entender como está estabelecida a estrutura de uma entrada no diretório.

Byte	Função
#00 a #07	Nome do arquivo
#08 a #0A	Extensão do arquivo
#0B	Byte de atributo
#0C a #15	Reservados
#16 e #17	Hora da criação do arquivo
#18 e #19	Data da criação do arquivo
#1A e #1B	Primeiro aglomerado do arquivo
#1C a #1F	Tamanho do arquivo

Tabela 6.3: Estrutura de uma entrada no diretório

As posições #1A e #1B armazenam o número do primeiro aglomerado do arquivo, mas, o que vem a ser isso? O sistema de armazenamento de arquivos no MS-DOS e no MSX-DOS utiliza a chamada **FAT** (do inglês **File Alocate Table** - Tabela de Alocação de Arquivos). A FAT nada mais é do que um mapa com a localização de todos os aglomerados que pertencem a um determinado arquivo. "Mas, o que é um aglomerado?" Um aglomerado é a menor partição lógica de um arquivo compreendida pelo sistema. No caso de disquetes de 5 1/4," um aglomerado equivale a um setor nos disquetes de face simples, e a dois setores nos disquetes de face dupla. Desta forma, por exemplo, se gravarmos um arquivo com, digamos, 20 bytes, gastaríamos 512 bytes num disquete de face simples e 1024 num disquete de face dupla. Na FAT, um aglomerado aponta para outro aglomerado, e este, por sua vez, aponta para outro, e assim por diante, até chegar a um aglomerado que aponta para o aglomerado #FFF, que, por sua vez, não existe. Assim, o sistema fica sabendo que o aglomerado que apontou para #FFF é o último do arquivo. Este é o funcionamento básico da FAT. Voltando à estrutura de uma entrada no diretório, temos nas posições #16 e #17 a hora da criação do arquivo, e nas posições #18 e #19 a data da criação dele. A codificação usada para esses valores é a mesma utilizada no FCB. De todas as posições na estrutura de diretório, a mais intrigante e, infelizmente, não usada no MSX-DOS é a posição #0B, que contém o chamado byte de atributo. Vejamos a sua codificação:

Bit	Função
0	Se estiver setado (em 1), indica que o arquivo em questão é somente para leitura, o que implica no arquivo não poder ser apagado ou modificado.
1	Se estiver setado (em 1), indica que o arquivo em questão não aparecerá na listagem de diretório promovida pelo comando DIR ou FILES.
2	Se estiver setado (em 1), indica que é um arquivo de sistema e como tal só poderá ser acessado pelo próprio sistema.
3	Se estiver setado (em 1), indica que o nome em questão não é de um arquivo, mas sim o rótulo (label) do disquete.
4	Se estiver setado (em 1), indica que o arquivo é na verdade, um subdiretório. Este é o único bit que o MSX-DOS reconhece, mas, mesmo assim, não aceita. Na listagem do diretório, o MSX-DOS coloca o sufixo DIR após o nome do arquivo.
5	Se estiver setado (em 1), indica que é um arquivo de dados, e não um programa.
6	Reservado
7	Reservado

Tabela 6.4: Estrutura do byte de atributo

Infelizmente, o MSX-DOS não utiliza nenhum dos recursos permitidos pelo byte de atributo. Portanto, não nos interessa estudá-lo em maiores detalhes. Vamos passar, agora, à análise de alguns endereços úteis da RAM.

ENDEREÇOS ÚTEIS DO BASIC DE DISCO

Existem alguns ponteiros para buffers na RAM que podem nos fornecer informações preciosas. Todos os ponteiros e endereços usados pelo sistema de disco se encontram acima do endereço #F200, sendo por este motivo muito úteis nos programas que utilizam o BASIC de Disco. Vale lembrar que os endereços abaixo são resultado de uma pesquisa pessoal (como foram quase todas as informações contidas neste e no próximo capítulo), de modo que não posso garantir que esses sejam todos os endereços usados pelo sistema de disco. Eventualmente, podem existir mais endereços com outras informações, mas creio que aquelas fornecidas pelos endereços aqui apresentados são mais do que suficientes.

Endereço #F247 : Este endereço contém o número do drive default. Se você quiser colocar o drive B como default, poderá entrar com a seguinte instrução em BASIC:

POKE &HF247,1

Se, depois, quiser retornar o drive A como default, entre com a seguinte instrução:

POKE &HF247,0

Endereço #F341 : Este endereço contém a identificação do slot (veja o Capítulo 5 para maiores esclarecimentos sobre o byte de identificação de slots) da página 0 (#0000 a #3FFF) da memória RAM.

Endereço #F342 : Este endereço contém a identificação do slot da página 1 (#4000 a #7FFF) da memória RAM.

Endereço #F343 : Este endereço contém a identificação do slot da página 2 (#8000 a #BFFF) da memória RAM.

Endereço #F344 : Este endereço contém a identificação do slot da página 3 (#C000 a #FFFF) da memória RAM.

Endereço #F346 : Este endereço indica se o disquete com que foi feito o boot continha ou não o sistema (MSXDOS.SYS). Se, no momento do boot, o sistema foi carregado, este endereço apresentará um valor diferente de #00, caso contrário, apresentará o valor #00. Suponha que você tenha dado o boot com um disquete sem sistema e, depois, queira carregar o sistema utilizando um outro disquete. Neste caso, o boot pelo primeiro disquete levou-o direto ao BASIC de Disco e inicializou este endereço com o valor #00. Desta forma, se você tentar o comando

CALL SYSTEM

o BASIC responderá com uma mensagem de erro. Mas, se você trocou o disquete sem sistema por um com sistema, poderá entrar com a seguinte seqüência de comandos para o sistema ser carregado:

POKE &HF346,1
CALL SYSTEM

Endereço #F347 : Este endereço indica o número de drives lógicos instalados no sistema. Se você só possuir um drive físico, mas no momento do boot não pressionou a tecla CONTROL, este endereço conterá o valor #02, indicando que existem #02 drives lógicos no sistema. Agora, se você pressionou a tecla CONTROL durante o boot, este endereço conterá o valor #01, indicando que só existe um drive lógico. Se você possuir dois drives físicos (A e B) conectados ao seu sistema, este endereço apresentará sempre o valor #02, a menos que antes do boot você desligue a alimentação do drive B e pressione a tecla CONTROL durante o boot.

Endereço #F348 : Este endereço contém a identificação do slot onde se encontra a interface do drive (veja o Capítulo 5 para maiores esclarecimentos sobre a identificação dos slots).

Endereço #F349 : Este endereço contém um ponteiro para uma área de 3 Kbytes com uma cópia da FAT (1,5 Kbytes) do drive B, seguida de uma cópia da FAT (1,5 Kbytes) do drive A.

Endereço #F34D : Este endereço contém um ponteiro para uma área de 1,5 Kbytes com uma cópia da FAT do disquete no drive default. Se você quiser saber o endereço inicial desta cópia da FAT, entre com o seguinte comando:

? HEX$(PEEK(&HF34D)+256*PEEK(&HF34E))

O resultado apresentado na tela indica o endereço inicial da cópia da FAT em RAM.

Endereço #F34F : Este endereço contém um ponteiro para uma área de 512 bytes usada como DMA do BASIC de Disco. Se você quiser saber o endereço inicial do DMA do BASIC de Disco, entre com o comando

? HEX$(PEEK(&HF34F)+256*PEEK(&HF350))

O resultado apresentado na tela indicará o endereço.

Endereço #F351 : Este endereço contém um ponteiro que aponta para uma área de 512 bytes na RAM, que é usada para a transferência de setores. Devido ao seu comprimento (512 bytes), a área apontada só pode receber um setor por vez. Se você quiser descobrir a localização do buffer de setores na RAM, entre com o seguinte comando em BASIC:

? HEX$(PEEK(&HF351)+256*PEEK(&HF352))

O resultado apresentado na tela será o endereço inicial do buffer dos setores.

Endereço #F353 : Este endereço contém um ponteiro que aponta para uma área de 37 bytes, usada pelo BASIC de Disco para armazenar o FCB do arquivo atual. Se você quiser descobrir o endereço inicial do FCB do arquivo atual, entre com o seguinte comando:

? HEX$(PEEK(&HF353)+256*PEEK(&HF354))

O resultado apresentado na tela será o endereço inicial do buffer dos setores.

Endereço #F355 : Este endereço contém um ponteiro que aponta para uma área de 21 bytes, que recebe o nome de **BPB**. O ponteiro neste endereço aponta para o BPB do drive A. Veja abaixo a descrição completa de todos os bytes desta área. Vale lembrar que as informações do BPB são atualizadas somente com o uso da função #1B do BDOS.

Endereço #F357 : Este endereço contém um ponteiro que aponta para o **BPB** do drive B (se existir). Permanecem válidas todas as observações feitas no endereço anterior.

Uma vez apresentados os endereços e o significado dos respectivos conteúdos, vamos para a análise da estrutura do BPB. Não tenho a menor idéia do porquê deste nome, mas o conteúdo deste buffer de 21 bytes é extremamente variado e útil, merecendo por isso mesmo um estudo mais detalhado.

A ESTRUTURA DO BPB (Bloco de Parâmetros da BIOS)

O BPB é um buffer de 21 bytes que armazena importantes informações sobre os disquetes. Vejamos o significado de cada byte ou par de bytes deste buffer.

Byte	Função
#00	Número do drive - 1 (0=A, 1=B)
#01	Tipo de formatação. Os valores usados no MSX-DOS e MS-DOS são: #F8 3 1/2" face simples #F9 3 1/2" face dupla #FC 5 1/4" face simples #FD 5 1/4" face dupla
#02 e #03	Número de bytes por setor
#04 e #05	Não consegui chegar a uma conclusão sobre o conteúdo destes endereços
#06	Número de faces - 1 (0=face simples, 1=face dupla)

Tabela 6.5: Estrutura do BPB

#07	Número de setores por aglomerado
#08 e #09	Número de setores reservados
#0A	Número de FATs
#0B	Número máximo de entradas no diretório
#0C e #0D	Número do primeiro setor da área de armazenamento (=número do setor final do diretório + 1)
#0E e #0F	Número total de aglomerados no disquete
#10	Número de setores por FAT
#11 e #12	Número do primeiro setor do diretório
#13 e #14	Endereço da cópia da FAT em RAM

Tabela 6.5: Estrutura do BPB(continuação)

Gostaria agora de apresentar alguns endereços úteis do MSX-DOS.

ENDEREÇOS ÚTEIS DO MSX-DOS

Existem alguns endereços do MSX-DOS que valem a pena mencionar. São eles:

Endereço #0000 : Este endereço contém um vetor para a partida a quente do MSX-DOS. Suponha que você queira sair de um programa e não tenha certeza dos estragos feitos na memória. Neste caso, seria conveniente sair com a instrução

JUMP#0000

que desviaria a execução para a rotina de partida a quente, e esta rotina, por sua vez, detectaria a necessidade ou não de se recarregar o sistema.

Endereço #0005 : Este endereço contém o vetor para o BDOS. Você pode usar o conteúdo dos endereços #0006 e #0007 (endereço do salto para o BDOS) para descobrir a maior posição possível para o stack pointer (SP). Para tanto, basta fazer o registro SP igual ao conteúdo -1 dos endereços #0006 e #0007.

Endereço #0030 : Vetor para a rotina CALSLT (chamada de rotinas entre slots).

Endereço #005C : Primeiro FCB do MSX-DOS.

Endereço #006C : Segundo FCB do MSX-DOS.

Endereço #0080 : Este endereço tem dupla função: armazena a linha de comando (a partir do endereço #0081) e serve como DMA para as transferências de e para o disquete. No caso da linha de comando, os caracteres lidos do teclado são armazenados a partir do endereço

#0081, contando somente aqueles após o nome do programa. No endereço #0080 é armazenado o número de caracteres após o nome do programa. Supondo que o usuário tenha entrado com a linha

WS CAPMSX-1.DOC

os caracteres armazenados a partir do endereço #0081 serão:

#0081	Espaço em branco
#0082	C
#0083	A
#0084	P
#0085	M
#0086	S
#0087	X
#0088	-
#0089	1
#008A	.
#008B	D
#008C	O
#008D	C

Quando utilizarmos este como o endereço inicial do DMA, devemos ter em mente que o comprimento total do DMA não poderá ultrapassar os 128 bytes, ou seja, o DMA deverá situar-se entre os endereços #0080 e #00FF, já que a partir do endereço #0100 temos o início do programa .COM carregado na memória.

Para completar a nossa viagem pelo sistema de disco, gostaria de apresentar três rotinas úteis: uma rotina para a leitura e gravação de setores, ativada pela BIOS do MSX; uma rotina para inicialização do BASIC de Disco; e uma rotina para provocar a parada do drive.

ALGUMAS ROTINAS ÚTEIS

Como você já sabe, todas as rotinas para o acesso ao drive estão contidas numa ROM de 16 Kbytes na interface do drive. Além de todas as rotinas/funções apresentadas acima, existem três rotinas que merecem destaque. Na Tabela 6.6 abaixo, temos os parâmetros para o acesso de tais rotinas.

Nome da rotina	: PHYDIO
Endereço inicial	: #0144
Modo de chamada	: CALL
Parâmetros de entrada	: HL=endereço da RAM a partir do qual serão colocados os setores lidos

Tabela 6.6: Rotinas úteis do sistema de disco

	DE =número do primeiro setor a ser lido B =número de setores a ler C =parâmetro de formatação do disquete. Este parâmetro obedece àqueles mostrados na posição #01 do BPB A =número do drive (0=A, 1=B, etc.) Flag de carry=se desejarmos efetuar uma leitura, esta flag deverá estar resetada; já se desejarmos efetuar uma gravação, devemos setá-la
Parâmetros de saída	: Flag de carry=se estiver setada, houve algum tipo de erro (de leitura ou gravação, dependendo do caso)
Registros alterados	: Todos
Nome da rotina	: INIBAS (dado por mim)
Endereço de entrada	: #4022 (no slot da interface do disco)
Modo de chamada	: Por intermédio de CALSLT
Parâmetros de entrada	: Nenhum
Parâmetros de saída	: Nenhum
Registros alterados	: Todos
Nome da rotina	: STPDRV (dado por mim)
Endereço inicial	: #4029 (no slot da interface do disco)
Modo de chamada	: Por intermédio de CALSLT
Parâmetros de entrada	: Nenhum
Parâmetros de saída	: Nenhum
Registros alterados	: Todos

Tabela 6.6: Rotinas úteis do sistema de disco(continuação)

A rotina PHYDIO é bem simples, e já foi inclusive usada por nós no programa 28 do Capítulo 5. A rotina INIBAS será bastante útil se você quiser promover uma volta ao BASIC a partir de um programa em execução no MSX-DOS. A rotina STPDRV (StopDrive) pára o motor do drive. Esta rotina é muito útil no início de programas como os jogos que desabilitam por comple-

to as interrupções, impedindo assim a parada do drive, que é feita através de um desvio do hook HTIMI.

Terminada a teoria, vamos às listagens de alguns programas que permitem uma ilustração do uso dos conceitos vistos anteriormente.

O PROGRAMA QUE IMPLEMENTA OS COMANDOS FINDFIRST E FINDNEXT

Este programa é uma continuação do programa de implementação de novos comandos, que viemos incrementando desde o Capítulo 3. A principal função deste programa é demonstrar o uso de duas funções do BDOS: a função #11, que procura o primeiro arquivo, e a função #12, que procura o arquivo seguinte. Vamos então às listagens.

Listagem em assembly Z-80 do código-fonte do programa que implementa os comandos FINDFIRST e FINDNEXT:

```
;programa que implementa os comandos FINDFIRST e FINDNEXT
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG29.BIN=PROG29.GEN
;
;onde PROG29.GEN é o nome do arquivo-texto com esta
;listagem

versao          equ     #002d
gtstck          equ     #00d5
gttrig          equ     #00d8
kilbuf          equ     #0156
linlen          equ     #f3b0
txtnam          equ     #f3b3
txtcgp          equ     #f3b7
calbas          equ     #0159
chrgtr          equ     #4666
evlexp          equ     #520e
getcrd          equ     #579c
varsrc          equ     #5ea4
anfile          equ     #6a0e
dridef          equ     #f247
dma             equ     #f34f
fcb             equ     #f353
bdos            equ     #f37d
crtcnt          equ     #f3b1
```

```
errflg          equ     #f414
valtyp          equ     #f663
savtxt          equ     #f6af
scrmod          equ     #fcaf
grpacx          equ     #fcb7
grpacy          equ     #fcb9
hkeyi           equ     #fd9a
herro           equ     #ffb1

        defb    #fe                     ;simula em CP/M
        defw    inicio                  ;o cabecalho de
        defw    fim-erro+rotina+#01
                                        ;um arquivo
        defw    inicio                  ;.BIN


                org     #d000

inicio
                di                      ;desabilita as interrupções
                in      a,(#a8)         ;lê a atual configuração dos
                ld      b,a             ;slots e prepara para ativar
                and     #f0             ;a página 1 do slot da RAM
                rrca                    ;
                rrca                    ;
                rrca                    ;
                rrca                    ;
                or      b               ;
                out     (#a8),a         ;ativa as páginas 1, 2 e 3 em
                and     #03             ;RAM e descobre o slot onde
                ld      (slot),a        ;se encontram os 64Kb de RAM
                ld      a,b             ;a=config. inicial dos slots
                ld      hl,novohook     ;promove o desvio do hook
                ld      de,herro        ;dos erros
                ld      bc,0005         ;
                ldir                    ;
                ld      hl,rotina       ;transfere a rotina para a
                ld      de,erro         ;página 1 da RAM
                ld      bc,fim-erro+#01 ;
                ldir                    ;
                out     (#a8),a         ;habilita a config. original
                ei                      ;habilita as interrupções
                ret                     ;retorna ao BASIC


novohook
                defb    #f7             ;RST #30
```

```
slot            defb    #00             ;identificação do slot
                defw    erro            ;endereço de desvio
                defb    #c9             ;RET


rotina
                org     #4000

erro
                push    af              ;salva os registros afetados
                push    bc              ;
                push    de              ;
                push    hl              ;
examerro        ld      a,e             ;a=código do erro
                cp      #02             ;é erro de sintaxe?
                jp      nz,aborta       ;Não, volta ao BASIC
pegachar        ld      hl,(savtxt)     ;Sim, obtém a posição do
                                        ;ponteiro do BASIC
                ld      a,(hl)          ;é final de linha?
                or      a               ;
                jp      nz,proxchar     ;Não, obtém o próximo carac.
pula            inc     hl              ;Sim, pula as 4 posições
                inc     hl              ;referentes ao número da
                inc     hl              ;linha e ao end. da próxima
                inc     hl              ;linha
proxchar        ld      ix,chrgtr       ;obtém o primeiro caractere
                call    chamabasic      ;do novo comando
compara         call    compstring      ;compara com a lista de novos
                                        ;comandos
aborta          pop     hl              ;recupera os registros salvos
                pop     de              ;
                pop     bc              ;
                pop     af              ;
                ret                     ;retorna ao BASIC


;a rotina compstring é a responsável pela localização da
;rotina do novo comando

compstring
                ld      (auxiliar),hl   ;salva o pont. do BASIC
                ld      hl,lista        ;hl aponta p/ lista de
                                        ;comandos
                ld      bc,endlista-lista
                                        ;bc=tamanho da lista
```

```
compstr1    ld      de,(auxiliar)    ;de aponta p/ comando em
                                     ;BASIC
            ld      a,(de)           ;obtém o primeiro byte
            cpir                     ;tenta encontrá-lo na
                                     ;lista
            jp      po,naoachei      ;se bc=0 -> não achou
compstr2    inc     de               ;achou o primeiro e
                                     ;parte
            ld      a,(de)           ;para a comparação dos
            cp      (hl)             ;demais caracteres. Vai
                                     ;para
            jp      nz,compstr1      ;compstr1 se achar um
                                     ;byte diferente.
            inc     hl               ;Caso contrário,
            ld      a,(hl)           ;continua até o fim do
                                     ;comando.
            cp      #00              ;Se chegou ao fim e,
            jp      z,achei          ;então, achou o comando
                                     ;na lista.
            jp      compstr2         ;Caso contrário, compara
naoachei    ret                      ;o próximo byte.
achei       inc     hl               ;Se achou, obtém o
                                     ;endereço
            ld      c,(hl)           ;de entrada da rotina,
            inc     hl               ;colocando-o em BC
            ld      b,(hl)           ;
            pop     af               ;retira da pilha o
                                     ;endereço de
            push    bc               ;retorno para a rotina
                                     ;erro e
                                     ;coloca na pilha o
                                     ;endereço de
                                     ;entrada da rotina do
                                     ;novo comando
            ld      (auxiliar),de    ;guarda o ponteiro
                                     ;do BASIC
            ex      de,hl            ;transfere o ponteiro
                                     ;para HL
            ret                      ;usa RET para dar um
                                     ;jump
                                     ;para o endereço contido
                                     ;BC que foi salvo na
                                     ;pilha
```

```
;a rotina lecomandos é a responsável pela interpretação dos
;valores que se seguem ao nome do novo comando. Os valores
;devem estar separados por vírgulas. Na entrada,o par IX
;aponta para uma tabela com valores default, e o registro B
;contém o número de valores a interpretar

lecomandos
                push    ix              ;salva o par IX
lecoman_1       push    ix              ;
                ld      ix,chrgtr       ;obtém um valor
                call    chamabasic
                pop     ix              ;recupera o ponteiro IX
                jr      z,lecoman_4     ;Se não houver valor, vai para
                                        ;lecoman_4
                cp      ","             ;é vírgula?
                jr      z,lecoman_3     ;Sim, assume o default
                dec     hl              ;Não, obtém o valor
                push    ix              ;salva o ponteiro
                push    bc              ;salva o contador
                ld      ix,evlexp       ;obtém o valor
                call    chamabasic      ;
                pop     bc              ;recupera o valor
                pop     ix              ;recupera o ponteiro
                ld      a,e             ;a=valor interpretado
                ld      (ix+#00),a      ;coloca-o na tabela
lecoman_3       inc     ix              ;incrementa o ponteiro
                djnz    lecoman_1       ;repete para os demais
                                        ;valores
lecoman_4       pop     ix              ;recupera o ponteiro salvo
                ret                     ;retorna


;a rotina erro_05 emite a mensagem de chamada ilegal

erro_05
                ld      a,#05           ;a=código do erro
                ld      (errflg),a      ;salva o erro em errflg
                jr      voltaerro       ;vai para voltaerro


;a rotina erro_13 emite a mensagem de tipo incompatível

erro_13
                ld      a,13            ;a=código do erro
```

```
            ld      (errflg),a          ;salva o erro em errflg
            jr      voltaerro           ;vai para voltaerro


;a rotina voltaerro substitui o erro original por um outro

voltaerro
            pop     hl                  ;recupera os pares
            pop     de                  ;salvos no início da
            pop     bc                  ;rotina erro
            pop     af                  ;
            ld      a,(errflg)          ;a=erro a ser emitido
            ld      e,a                 ;e=erro a ser emitido
            ret                         ;volta para a rotina
                                        ;de manipulação de erros
                                        ;do interpretador BASIC


;a rotina volta promove a volta ao BASIC sem que haja
;interrupção no processamento, ou seja, sem que o sistema
;detecte o erro ocorrido

volta
            dec     hl                  ;hl aponta para o final
                                        ;do último comando
            xor     a                   ;modifica para nenhum erro
            ld      (errflg),a          ;ocorrido
            pop     de                  ;recupera os registros
            pop     de                  ;salvos no início da rotina
            pop     bc                  ;erro. Note que hl não é
            pop     af                  ;recuperado.
            pop     de                  ;obtém o end. de retorno da
                                        ;nossa rotina para a rotina
                                        ;e chaveamento de slots
            pop     bc                  ;obtém o end. de retorno da
                                        ;rotina de chaveamento de
                                        ;slots para RST #30
            pop     af                  ;obtém o end. de retorno de
                                        ;RST #30 para o hook
            pop     ix                  ;obtém/destrói o end. de
                                        ;retorno do hook para a
                                        ;rotina
                                        ;de erro do interpretador
                                        ;BASIC
            pop     ix                  ;obtém/destrói o end. de
```

```
                                        ;retorno
                                        ;da rotina de erro para o
                                        ;interpretador BASIC
              push    af                ;repõe na pilha os end. de
              push    bc                ;retorno salvos nestes
              push    de                ;registros
              ld      ix,chrgtr         ;faz com que hl aponte
              jp      chamabasic        ;para o próximo comando
              ret                       ;e retorna ao BASIC


chamabasic
              call    calbas            ;chama a rotina calbas
              ret                       ;retorna

inverte
              ld      b,#04             ;b=núm. de valores
              ld      ix,tabinver       ;ix=tabela default
              call    lecomandos        ;lê os 4 valores
              ld      (auxiliar),hl     ;salva ptr. do BASIC
              call    rotinversao       ;chama a rotina de
                                        ;inversão
              ld      hl,(auxiliar)     ;recupera ptr. do BASIC
              jp      volta             ;volta para o BASIC


rotinversao
              ld      a,(ix+#00)        ;obtém coord. x
              ld      (grpacx),a        ;salva em grpacx
              ld      a,(ix+#01)        ;obtém coord. y
              ld      (grpacy),a        ;salva em grpacy
              ld      a,(ix+#02)        ;obtém o comp. da string
              ld      (compstr),a       ;salva em compstr
              cp      33                ;é >= 33?
              jp      nc,erroinv_05     ;Sim, chamada ilegal
              ld      a,(ix+#03)        ;a=flag de recuperação
              or      a                 ;está setada?
              jr      z,inverte0        ;Não, vai para inverte0
              ld      a,(inversao)      ;Sim, já foi feita alguma
              or      a                 ;inversão?
              jr      z,inverte0        ;Não, vai para inverte0
              ld      a,(bufnor)        ;Sim, obtém o comprimento
                                        ;da string.
              or      a                 ;é zero?
              jr      z,inverte0        ;Sim, vai para inverte0
              ld      hl,(bufnor+#01)   ;Não, obtém a pos. da string
              call    setvdpwt          ;antiga e prepara o VDP
```

```
          ld      hl,bufnor+#03     ;hl->início da string antiga
          ld      a,(bufnor)        ;a=comp. da string antiga
          ld      b,a               ;a=comp. da string antiga
          call    esclinha          ;escreve a string antiga
                                    ;no modo normal
inverte0
          ld      hl,(auxiliar)     ;hl=ponteiro do BASIC
          ld      a,(compstr)       ;a=comp. da string
          or      a                 ;comprimento=0?
          jp      z,voltainv        ;Sim, volta sem erro
          ld      ix,bufnor         ;Não, ajusta os buffers
          ld      iy,bufinv         ;para a string normal e
          ld      (ix+#00),a        ;para a string invertida
          ld      (iy+#00),a        ;com o comprimento da
          call    calpadvid         ;calcula os padrões do vídeo
          ld      a,(grpacx)        ;obtém a coord. x
          ld      e,a               ;coloca em e
          ld      a,(colunas)       ;obtém o número de colunas
          cp      e                 ;compara com o valor lido
          jp      c,erroinv_05      ;Se valor lido > colunas,
                                    ;volta com erro
          ld      a,(grpacy)        ;obtém a coord. y
          cp      24                ;é maior do que 23?
          jp      nc,erroinv_05     ;Sim, volta com erro
          call    lefrase           ;lê a string a inverter
inverte3  di                        ;desabilita as interrupções
          ld      hl,(tabpad)       ;hl=end. da tab. de padrões
          push    hl                ;salva o end. da tab. de
                                    ;padroes
          ld      de,#e0*8          ;calcula o endereço do
          add     hl,de             ;desenho do caractere #e0
          ex      de,hl             ;de=end. do des. do carac. #e0
          ld      c,#e0             ;c=#e0
          ld      b,(ix+#00)        ;b=número de carac. da string
          ld      ix,bufnor+#03     ;ix=end. da string normal
          ld      iy,bufinv+#03     ;iy=end. da string invertida
          pop     hl                ;recupera hl
loopinv1  push    bc                ;salva bc
          push    hl                ;salva hl
          push    de                ;salva de
          ld      de,#0008          ;bytes para o desenho de cada
                                    ;caractere
          ld      b,(ix+#00)        ;b=caractere a inverter
loopinv2  add     hl,de             ;calcula o endereço desse
          djnz    loopinv2          ;caractere
          pop     de                ;recupera de
          ld      b,#08             ;prepara a modificação
```

```
loopinv3    call    rdvram          ;lê o byte original
            cpl                     ;inverte
            ex      de,hl           ;transfere para o novo
            call    wtvram          ;destino
            ex      de,hl           ;
            inc     hl              ;hl=hl+1
            inc     de              ;de=de+1
            djnz    loopinv3        ;b=b-1
            pop     hl              ;recupera hl
            pop     bc              ;recupera bc
            ld      (iy+#00),c      ;modifica a string
            inc     c               ;c=c+1
            inc     ix              ;aponta para o próximo carac.
            inc     iy
            djnz    loopinv1        ;repete até o fim da string
envinvert
            ld      iy,bufinv       ;recupera o ponteiro do
                                    ;buffer
            ld      l,(iy+#01)      ;hl=end. de escrita da string
            ld      h,(iy+#02)      ;
            call    setvdpwt        ;ajusta o VDP para escrita
            ld      b,(iy+#00)      ;b=núm. de carac. da string
            ld      hl,bufinv+#03   ;end. da string
            call    esclinha        ;escreve a frase invertida
            ld      a,#ff           ;sinaliza uma inversão
            ld      (inversao),a    ;
            ei                      ;habilita as interrupções
            ret                     ;retorna

erroinv_05
            pop     af              ;destrói o end. de retorno
            jp      erro_05         ;vai para erro_04

voltainv
            pop     af              ;destrói a pilha
            jp      volta           ;volta para o BASIC


tabinver
            defb    #00             ;posição x
            defb    #00             ;posição y
            defb    #00             ;comprimento
            defb    #00             ;flag de restauração


newcls
            ld      b,#02           ;b=núm. de valores
```

```
        ld      ix,tabnewcls        ;ix->tab. de valores default
        call    lecomandos          ;
        ld      (auxiliar),hl       ;salva o ponteiro do BASIC
        ld      a,(ix+#00)          ;obtém o primeiro valor
        cp      #02                 ;verifica o limite
        jp      nc,erro_05          ;se estiver fora, volta com
                                    ;erro
        call    calpadvid           ;calcula os padrões do vídeo
        ld      a,(colunas)         ;obtém o número de colunas
        ld      e,a                 ;e=número de colunas
        ld      a,(ix+#01)          ;a=número de rotações
        inc     e
        cp      e                   ;rotações>colunas?
        jp      nc,erro_05          ;Sim, volta com erro
        or      a                   ;Zero rotações?
        jp      z,volta             ;Sim, volta sem erro
        ld      a,(ix+#00)          ;a=tipo de rotação
        or      a                   ;é zero (esquerda)?
        jp      nz,newcls_0         ;Não, vai para newcls_0


;newcls_1 faz o cls com rotação para a esquerda

newcls_1

        di                          ;desabilita as interrupções
        ld      b,(ix+#01)          ;b=número de rotações
loopcls_11
        push    bc                  ;salva o contador externo
        ld      hl,(txtnam)         ;hl=end. tabela de nomes
        ld      a,(colunas)         ;a=núm. de colunas
        ld      e,a                 ;de=núm. de colunas
        ld      d,#00               ;
        ld      a,(crtcnt)          ;a=número de linhas
        ld      b,a                 ;b=número de linhas
loopcls_12
        push    bc                  ;salva contador intermediário
        call    setvdprd            ;prepara o VDP para leitura
        ld      a,(colunas)         ;a=núm. de colunas
        push    de                  ;salva de
        push    hl                  ;salva hl
        ld      hl,bufferlinha      ;hl aponta para o buffer
        ld      b,a                 ;b=núm. de colunas
        call    lelinha             ;lê uma linha
        ld      a,#20               ;a=carac. espaço em branco
        ld      (hl),a              ;coloca o espaço na últ. pos.
```

```
                pop     hl                  ;recupera hl
                pop     de                  ;recupera de
                call    setvdpwt            ;prepara o VDP para escrita
                push    de                  ;salva de
                push    hl                  ;salva hl
                ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
                ld      a,(colunas)         ;a=núm. de colunas
                ld      b,a                 ;b=núm. de colunas
                call    esclinha            ;envia a linha já rotada
                pop     hl                  ;recupera hl
                pop     de                  ;recupera de
                add     hl,de               ;hl aponta para a próx. linha
                pop     bc                  ;recupera bc
                djnz    loopcls_12          ;repete para as demais linhas
                pop     bc                  ;recupera bc
                djnz    loopcls_11          ;repete até terminar todas
                                            ;as rotações
                ei                          ;habilita as interrupções
                ld      hl,(auxiliar)       ;recupera o ponteiro do BASIC
                jp      volta               ;retorna ao BASIC


;newcls_0 faz o cls com rotação para a direita

newcls_0
                di                          ;desabilita as interrupções
                ld      b,(ix+#01)          ;b=número de rotações
loopcls_01
                push    bc                  ;salva contador externo
                ld      hl,(txtnam)         ;hl=end. tabela de nomes
                ld      a,(colunas)         ;a=núm. de colunas
                ld      e,a                 ;de=núm. de colunas
                ld      d,#00               ;
                ld      a,(crtcnt)          ;a=número de linhas
                ld      b,a                 ;b=número de linhas
loopcls_02
                push    bc                  ;salva contador intermediário
                call    setvdprd            ;prepara o VDP para leitura
                ld      a,(colunas)         ;a=núm. de colunas
                push    de                  ;salva de
                push    hl                  ;salva hl
                ld      hl,bufferlinha+1    ;hl aponta para o buffer+1
                ld      b,a                 ;b=núm. de colunas
                call    lelinha             ;lê uma linha
                ld      a,#20               ;a=carac. espaço em branco
                ld      hl,bufferlinha      ;hl aponta para o buffer
```

```
        ld      (hl),a          ;coloca o espaço na prim.
                                ;posição
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        call    setvdpwt        ;prepara o VDP para escrita
        push    de              ;salva de
        push    hl              ;salva hl
        ld      hl,bufferlinha  ;hl aponta para o buffer
        ld      a,(colunas)     ;a=núm. de colunas
        ld      b,a             ;b=núm. de colunas
        call    esclinha        ;envia a linha já rotada
        pop     hl              ;recupera hl
        pop     de              ;recupera de
        add     hl,de           ;hl aponta para a próx. linha
        pop     bc              ;recupera bc
        djnz    loopcls_02      ;repete para as demais linhas
        pop     bc              ;recupera bc
        djnz    loopcls_01      ;repete até terminar todas
                                ;as rotações
        ei                      ;habilita as interrupções
        ld      hl,(auxiliar)   ;recupera o ponteiro do BASIC
        jp      volta           ;retorna ao BASIC


tabnewcls
        defb    #00             ;tipo de cls
        defb    #00             ;número de rotações


window
        ld      ix,tabwindow    ;ix aponta p/ tabela
        ld      b,#05           ;b=núm. de comandos
        call    lecomandos      ;lê as coordenadas
        ld      (auxiliar),hl   ;salva o ptr. do BASIC
        call    calpadvid       ;calcula os parâmetros
                                ;do vídeo
        ld      ix,tabwindow    ;ix aponta p/ tabela
        ld      h,(ix+#00)      ;h=x1
        ld      l,(ix+#01)      ;l=y1
        ld      d,(ix+#02)      ;d=x2
        ld      e,(ix+#03)      ;e=y2
        ld      a,h             ;testa se x2>x1
        cp      d               ;
        jp      nc,erro_05      ;x2<=x1->erro
        ld      a,l             ;testa se y2>y1
```

```
            cp      e                   ;
            jp      nc,erro_05          ;y2<=y1->erro
            ld      a,e                 ;a=y2
            cp      24                  ;y2>=24?
            jp      nc,erro_05          ;Sim, volta com erro
            ld      a,(colunas)         ;a=colunas na tela
            dec     a
            cp      d                   ;x2>=colunas?
            jp      c,erro_05           ;Sim, volta com erro
            ld      a,(ix+#04)          ;a=tipo de apresentação
            or      a                   ;é zero?
            jp      nz,zoom             ;Não, vai para zoom
            call    janela              ;Sim, chama janela
windvolta
            ld      hl,(auxiliar)       ;hl=ponteiro do BASIC
            jp      volta               ;volta para o BASIC


;A rotina zoom cria o efeito de explosão da janela

zoom
            xor     a                   ;zera o acumulador
            ld      (bufcol),a          ;zera o buffer da coluna
            ld      (buflin),a          ;zera o buffer da linha
            ld      (coror1),hl         ;salva as coordenadas x1,y1
            ld      (coror2),de         ;salva as coordenadas x2,y2
            ld      a,h                 ;carrega a com h (x1)
            add     a,d                 ;soma com d (x2) e divide
            srl     a                   ;por dois para achar Xmedio
            dec     a                   ;decrementa Xmedio
            ld      h,a                 ;carrega h (x1) com Xmedio
            inc     a                   ;incrementa o ponto médio
                                        ;para que
            inc     a                   ;x2=x1+1, onde x1=Xmedio-1
            ld      d,a                 ;carrega d com x2
            ld      a,l                 ;a=y1
            add     a,e                 ;soma com e (y2) e divide
            srl     a                   ;por dois para achar Ymedio
            dec     a                   ;decrementa Ymedio
            ld      l,a                 ;carrega l (y1) com Ymedio
            inc     a                   ;incrementa o ponto médio
                                        ;para que
            inc     a                   ;y2=y1+1, onde y1=Ymedio-1
            ld      e,a                 ;carrega e com y2
            call    janela              ;desenha a primeira janela
```

```
looptest     call    coluna          ;verifica em rel. à coluna-
                                     ;limite
             call    linha           ;verifica em rel. à linha-
                                     ;limite
             call    janela          ;desenha a janela
             call    espjanela       ;retardo para tornar o efeito
                                     ;visível
             call    teste2          ;verifica se já chegou à
                                     ;janela final
             jr      nz,looptest     ;Não, continua com a explosão
             ld      hl,(coror1)     ;imprime a janela final
             ld      de,(coror2)     ;
             call    janela          ;
             jp      windvolta       ;retorna ao BASIC


coluna       ld      a,(coror1+#01)  ;a=coord. x1 original
             cp      h               ;já foi atingida?
             jr      nc,fimcol       ;Sim, vai para fimcol
             dec     h               ;Não, x1=x1-1
             inc     d               ;x2=x2+1
             ret                     ;retorna
fimcol       ld      a,#01           ;indica que a coluna-limite
             ld      (bufcol),a      ;foi atingida
             ret                     ;retorna

linha        ld      a,(coror1)      ;a=coord. y1 original
             cp      l               ;já foi atingida?
             jr      nc,fimlin       ;Sim, vai para fimlin
             dec     l               ;Não, y1=y1-1
             inc     e               ;y2=y2+1
             ret                     ;retorna
fimlin       ld      a,#01           ;indica que a linha-limite
             ld      (buflin),a      ;foi atingida
             ret                     ;retorna

espjanela    push    af              ;salva os registros
             push    hl              ;afetados
             ld      hl,#1000        ;altere este valor para mudar
espjanela1
             dec     hl              ;o retardo. Decrementa hl
             ld      a,h             ;verifica se chegou
             or      l               ;ao fim
             jr      nz,espjanela1   ;Não, volta para espjanela1
             pop     hl              ;recupera os registros
             pop     af              ;
```

```
                ret                        ;e retorna

teste2          ld      a,(bufcol)         ;verifica se a coluna-
                cp      #01                ;limite foi atingida
                jr      z,contes           ;Sim, vai para contes
                ret                        ;Não, retorna
contes          ld      a,(buflin)         ;verifica se a linha-
                cp      #01                ;limite foi atingida
                ret                        ;retorna. Flag Z será o
                                           ;indicador.


;a rotina janela é a responsável pelo desenho da própria
;janela no vídeo

janela
                push    bc                 ;salva todos os registros
                push    de                 ;alterados pela rotina
                push    hl                 ;
                call    posit              ;posiciona o cursor em x1,y1
                ld      a,d                ;a=x2
                sub     h                  ;a=x2-x1
                dec     a                  ;a=núm. de pos. da linha do
                                           ;topo
                ld      b,a                ;b=núm. de pos. da linha do
                                           ;topo
                push    bc                 ;salva núm. pos. da linha do
                                           ;topo
                ld      a,#18              ;a=carac. do canto sup. esq.
                out     (#98),a            ;escreve o caractere
                ex      (sp),hl            ;demora para sincronização
                ex      (sp),hl            ;
janel1          ld      a,#17              ;a=traço horizontal
                out     (#98),a            ;escreve a linha superior
                ex      (sp),hl            ;demora para sincronização
                ex      (sp),hl            ;
                djnz    janel1             ;
                ld      a,#19              ;a=carac. do canto sup. dir.
                out     (#98),a            ;escreve o caractere
                pop     bc                 ;recupera núm. pos. linha do
                                           ;topo
                ld      l,e                ;l=y2
                call    posit              ;coloca o cursor em x1,y2
                ld      a,#1a              ;a=carac do canto inf.
                                           ;esquerdo
                out     (#98),a            ;escreve o caractere
```

```
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
janel2          ld      a,#17           ;a=traço horizontal
                out     (#98),a         ;escreve a linha inferior
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
                djnz    janel2          ;
                ld      a,#1b           ;a=carac. do canto inf. dir.
                out     (#98),a         ;escreve o caractere
                pop     hl              ;recupera x1,y1
                pop     de              ;recupera x2,y2
                push    de              ;salva x2,y2
                push    hl              ;salva x1,y1
                ld      a,e             ;a=y2
                sub     l               ;a=y2-y1
                dec     a               ;a=núm. de pos. verticais
                ld      b,a             ;b=núm. de pos. verticais
                ld      a,d             ;a=x2
                sub     h               ;a=x2-x1
                dec     a               ;a=núm. de pos. horizontais
                ld      c,a             ;c=núm. de pos. horizontais
                inc     l               ;y1=y1+1
janel3          call    posit           ;posiciona o cursor
                ld      a,#16           ;a=traço vertical
                out     (#98),a         ;escreve o caractere
                push    bc              ;salva núm. de pos. verticais
                ld      b,c             ;b=núm. de pos. horizontais
janel4          ld      a,#20           ;a=caractere espaço em branco
                out     (#98),a         ;escreve o carac. para limpar
                ex      (sp),hl         ;demora para sincronização
                ex      (sp),hl         ;
                djnz    janel4          ;a janela
                ld      a,#16           ;a=traço vertical
                out     (#98),a         ;escreve o caractere
                inc     l               ;y1=y1+1
                pop     bc              ;recupera núm. de pos.
                                        ;verticais
                djnz    janel3          ;repete até acabar as linhas
                                        ;vert.
                pop     hl              ;recupera os registros salvos
                pop     de              ;
                pop     bc              ;
                ret                     ;e retorna


tabwindow
                defb    #00             ;coordenada x1
```

```
                defb    #00                 ;coordenada y1
                defb    #00                 ;coordenada x2
                defb    #00                 ;coordenada y2
                defb    #00                 ;tipo de apresentação
bufcol          defb    #00                 ;buffer auxiliar
buflin          defb    #00                 ;buffer auxiliar
coror1          defw    #0000               ;buffer auxiliar
coror2          defw    #0000               ;buffer auxiliar


;a rotina menu faz a seleção em menus

menu
                ld      ix,tabmenu          ;ix aponta p/ tabela
                ld      b,#06               ;b=núm. de argumentos
                call    lecomandos          ;lê os argumentos
                dec     hl                  ;decrementa o ptr. do BASIC
                ld      ix,chrgtr           ;obtém o próximo caractere
                call    chamabasic          ;
                cp      ","                 ;é vírgula?
                jp      nz,aborta           ;Não, erro de sintaxe
                ld      ix,chrgtr           ;Sim, obtém o próximo
                call    chamabasic          ;caractere
                jp      z,aborta            ;Se for fim de linha->aborta
                ld      ix,varsrc           ;procura a variável
                call    chamabasic          ;inteira
                ld      (endvar),de         ;salva o end. da var.
                ld      a,(valtyp)          ;obtém o tipo da var.
                cp      #02                 ;é inteiro?
                jp      nz,erro_13          ;Não, erro de tipo
                ld      (auxiliar),hl       ;salva o ptr. do BASIC
                ld      ix,tabinver         ;ix aponta p/ tabinver
                ld      iy,tabmenu          ;iy aponta p/ tabmenu
                ld      a,(iy+#02)          ;a=x2
                cp      (iy+#00)            ;x2>=x1?
                jp      c,erro_05           ;Não, emite erro
                ld      a,(iy+#03)          ;a=y2
                cp      (iy+#01)            ;y2>=y1?
                jp      c,erro_05           ;Não, emite erro
                call    calpadvid           ;calcula pars. do vídeo
                ld      a,(colunas)         ;a=x1
                dec     a
                cp      (iy+#00)            ;x1>colunas?
                jp      c,erro_05           ;Sim, emite erro
                cp      (iy+#02)            ;x2>colunas?
```

```
            jp      c,erro_05         ;Sim, emite erro
            ld      a,24              ;a=núm. máx. de linhas
            cp      (iy+#01)          ;y1>24?
            jp      c,erro_05         ;Sim, emite erro
            cp      (iy+#03)          ;y2>24?
            jp      c,erro_05         ;Sim, emite erro
            ld      h,(iy+#00)        ;h=x1
            ld      l,(iy+#01)        ;l=y1
            ld      d,(iy+#02)        ;d=x2
            ld      e,(iy+#03)        ;e=y2
            ld      a,(iy+#04)        ;a=comprimento das opções
            ld      (ix+#02),a        ;salva em tabinver
            ld      a,#01             ;a=#01
            ld      (ix+#03),a        ;ajusta a flag de
                                      ;restauração
            ld      (cooratual),hl    ;salva a pos. atual
            xor     a                 ;zera a opção atual
            ld      (opcao),a         ;
            ld      (inversao),a      ;zera a inversão
            ld      a,h               ;compara x1 com
            cp      d                 ;x2
            jp      z,vertical        ;se x1=x2 mov. vertical
            ld      a,l               ;compara y1 com
            cp      e                 ;y2
            jp      nz,aborta         ;se não forem iguais
                                      ;emite erro de sintaxe


;a rotina horizontal controla o mov. horizontal. Nesta rotina,
;temos o cálculo do valor da opção final e, a
;partir de horizont_2, temos o controle do movimento
;propriamente dito

horizontal
            ld      a,(compopcao)     ;a=comp. das opções
            ld      e,a               ;salva a em e
            ld      a,(passo)         ;a=incremento entre as
                                      ;opções
            add     a,e               ;a=passo+comprimento
            ld      e,a               ;salva a em e
            ld      a,(coordx1)       ;a=x1
            ld      c,a               ;c=x1
            ld      a,(coordx2)       ;a=x2
            ld      b,#00             ;zera o registro b
horizont_1
            sub     e                 ;subtrai a de e
            jr      c,horiz_11        ;Se estourou, sai do loop
```

```
                inc     b                  ;incrementa b
                cp      c                  ;a=x1?
                jr      nz,horizont_1      ;Não, volta para horizont_1
horiz_11        ld      a,b                ;a=opção máxima
                ld      (opcaofim),a       ;salva em opcaofim

horizont_2
                ld      ix,tabinver        ;ix ponta p/ tabinver
                ld      hl,(cooratual)     ;hl=posição atual
                ld      a,h                ;
                ld      (ix+#00),a         ;
                ld      a,l                ;
                ld      (ix+#01),a         ;
                call    rotinversao        ;faz a inversão
                call    rdstick            ;lê o joystick/teclado
                cp      #03                ;o mov. foi para a direita?
                jp      z,movdir           ;Sim, vai para movdir
                cp      #07                ;o mov. foi para a esquerda?
                jp      z,movesq           ;Sim, vai para movesq
                cp      #ff                ;Return ou tiro?
                jp      z,fimsel           ;Sim, vai para fimsel
                jr      horizont_2         ;fecha o loop


;a rotina movdir controla o movimento para a direita.

movdir
                ld      a,(passo)          ;a=incremento
                ld      e,a                ;e=a
                ld      a,(xatual)         ;a=posição x atual
                add     a,e                ;soma x atual com o
                                           ;incremento
                ld      e,a                ;e=x atual+passo
                ld      a,(compopcao)      ;a=comp. das opções
                add     a,e                ;a=x atual+passo+comp.
                ld      e,a                ;e=x calculado
                ld      a,(coordx2)        ;a=coord. x limite
                cp      e                  ;x calculado > x limite?
                jr      c,movdir_1         ;Sim, vai para movdir_1
                ld      a,e                ;Não, a=x calculado
                ld      (xatual),a         ;modifica x atual
                ld      a,(opcao)          ;incrementa o valor da
                inc     a                  ;opção
                ld      (opcao),a          ;
                jp      horizont_2         ;volta para horizont_2
movdir_1
```

```
                ld      a,(coordx1)     ;hl=coord. original
                ld      h,a             ;
                ld      a,(coordy1)     ;
                ld      l,a             ;
                ld      (cooratual),hl  ;salva em cooratual
                xor     a               ;zera a opção
                ld      (opcao),a       ;salva em opção
                jp      horizont_2      ;volta para horizont_2

;a rotina movesq controla o movimento para a esquerda

movesq
                ld      a,(passo)       ;a=incremento
                ld      e,a             ;e=a
                ld      a,(xatual)      ;a=x atual
                ld      c,a             ;salva a em e
                ld      a,(coordx1)     ;a=x1
                cp      c               ;x atual=x1
                jr      z,movesq_1      ;Sim, vai para movesq_1
                ld      a,c             ;Não, calcula nova pos.
                sub     e               ;a=x atual-passo
                ld      e,a             ;e=x atual-passo
                ld      a,(compopcao)   ;a=comp. das opções
                ld      c,a             ;salva a em c
                ld      a,e             ;a=x atual-passo
                sub     c               ;a=x atual-passo-comp
                ld      (xatual),a      ;modifica xatual
                ld      a,(opcao)       ;decrementa o valor
                dec     a               ;da opção
                ld      (opcao),a       ;
                jp      horizont_2      ;volta para horizont_2

movesq_1
                ld      a,(coordx2)     ;hl=coord. original final
                ld      h,a             ;
                ld      a,(coordy2)     ;
                ld      l,a             ;
                ld      (cooratual),hl  ;salva em cooratual
                ld      a,(opcaofim)    ;a=opção final
                ld      (opcao),a       ;salva em opcao
                jp      horizont_2      ;fecha o loop


;a rotina vertical controla o movimento vertical. Nesta rotina,
;temos o cálculo do valor da opção final e, a
;partir de vertical_2, temos o controle do movimento
;propriamente dito
```

```
vertical
            ld      a,(passo)       ;a=incremento entre as
                                    ;opções
            ld      e,a             ;salva a em e
            ld      a,(coordy1)     ;a=y1
            ld      c,a             ;c=y1
            ld      a,(coordy2)     ;a=y2
            ld      b,#00           ;zera o registro b
vertical_1
            sub     e               ;a=a-passo
            inc     b               ;incrementa o registro b
            cp      c               ;a=y1?
            jr      nz,vertical_1   ;Não, volta para vertical_1
            ld      a,b             ;a=opção máxima
            ld      (opcaofim),a    ;salva em opcaofim

vertical_2
            ld      ix,tabinver     ;ix aponta p/ tabinver
            ld      hl,(cooratual)  ;hl=posição atual
            ld      a,h             ;salva a pos. a inverter
            ld      (ix+#00),a      ;
            ld      a,l             ;
            ld      (ix+#01),a      ;
            call    rotinversao     ;faz a inversão
            call    rdstick         ;lê o joystick/teclado
            cp      #05             ;o movimento foi para baixo?
            jp      z,movbai        ;Sim, vai para movbai
            cp      #01             ;o movimento foi para cima?
            jp      z,movcim        ;Sim, vai para movcim
            cp      #ff             ;Return ou tiro?
            jp      z,fimsel        ;Sim, vai para fimsel
            jp      vertical_2      ;Não, fecha o loop


;a rotina movbai controla o movimento para baixo.

movbai
            ld      a,(passo)       ;a=incremento entre as
                                    ;opções
            ld      e,a             ;salva a em e
            ld      a,(yatual)      ;a=y atual
            add     a,e             ;a=y atual+passo
            ld      e,a             ;salva o resultado em e
            ld      a,(coordy2)     ;a=y2
            cp      e               ;y atual>y2?
            jp      c,movbai_1      ;Sim, vai para movbai_1
```

```
          ld      a,e              ;Não, a=y atual
          ld      (yatual),a       ;salva em yatual
          ld      a,(opcao)        ;incrementa a opção
          inc     a                ;atual
          ld      (opcao),a        ;salva em opção
          jp      vertical_2       ;volta para vertical_2

movbai_1
          ld      a,(coordx1)      ;hl=posição inicial
          ld      h,a              ;
          ld      a,(coordy1)      ;
          ld      l,a              ;
          ld      (cooratual),hl   ;salva em cooratual
          xor     a                ;zera a opção atual
          ld      (opcao),a        ;
          jp      vertical_2       ;volta para vertical_2


;a rotina movcim controla o movimento para cima

movcim
          ld      a,(passo)        ;a=incremento entre as
                                   ;opções
          ld      e,a              ;salva a em e
          ld      a,(coordy1)      ;a=y1
          ld      c,a              ;salva y1 em c
          ld      a,(yatual)       ;a=y atual
          cp      c                ;y atual=y1?
          jr      z,movcim_1       ;Sim, vai para movcim_1
          sub     e                ;Não, a=y atual-passo
          ld      (yatual),a       ;salva em yatual
          ld      a,(opcao)        ;decrementa o valor da
          dec     a                ;opção atual
          ld      (opcao),a        ;salva em opção
          jp      vertical_2       ;volta para vertical_2

movcim_1
          ld      a,(coordx2)      ;hl=posição final
          ld      h,a              ;
          ld      a,(coordy2)      ;
          ld      l,a              ;
          ld      (cooratual),hl   ;salva em cooratual
          ld      a,(opcaofim)     ;a=valor da opção final
          ld      (opcao),a        ;salva em opção
          jp      vertical_2       ;volta para vertical_2
```

```
;a rotina fimsel promove a atualização da variável inteira
;e retorna para o BASIC

fimsel
                ld      hl,(endvar)     ;hl=end. da var. inteira
                ld      a,(opcao)       ;a=valor da opção
                ld      (hl),a          ;atualiza a variável
                inc     hl              ;
                ld      (hl),#00        ;
                call    kilbuf          ;limpa o buffer do teclado
                ld      hl,(auxiliar)   ;obtém o ptr. do BASIC
                jp      volta           ;volta para o BASIC


tabmenu
coord1
coordx1         defb    #00
coordy1         defb    #00

coord2
coordx2         defb    #00
coordy2         defb    #00

compopcao       defb    #00

passo           defb    #00


;a rotina findfirst encontra o primeiro arquivo
;que se corresponda com a especificação passada
;pelo BASIC

findfirst
                ld      a,#11           ;a=função do BDOS
                ld      (comfind),a     ;salva em comfind
                jr      comandfind      ;salta para comandfind


;a rotina findnext encontra o próximo arquivo
;que se corresponda com a especificação passada
;pelo BASIC

findnext
                ld      a,#12           ;a=função do BASIC
                ld      (comfind),a     ;salva em comfind
```

```
;a rotina comandfind executa a procura do
;primeiro e do próximo arquivo de acordo
;com o valor em comfind.

comandfind
            ld      ix,chrgtr        ;obtém o primeiro
            call    chamabasic       ;argumento
            jp      z,aborta         ;se não for caractere,
            jp      c,aborta         ;aborta
            ld      ix,varsrc        ;procura a variável
            call    chamabasic       ;
            ld      (endvar),de      ;salva o end. da variável
            ld      a,(valtyp)       ;verifica se é string
            cp      #03              ;
            jp      nz,erro_13       ;Não, emite type mismatch
            xor     a                ;inicializa a variável
            ld      (de),a           ;com tamanho
            inc     de               ;e endereço inicial
            ld      (de),a           ;iguais a zero
            inc     de               ;
            ld      (de),a           ;
            dec     hl               ;
            ld      ix,chrgtr        ;obtém o próximo caractere
            call    chamabasic       ;
            cp      ","              ;é vírgula?
            jp      nz,aborta        ;Não, erro de sintaxe
            ld      ix,chrgtr        ;obtém o próximo caractere
            call    chamabasic       ;
            cp      #22              ;são aspas?
            jp      nz,aborta        ;Não, erro de sintaxe
            ld      ix,anfile        ;obtém o nome do arquivo
            call    chamabasic       ;
            ld      (auxiliar),hl    ;salva o ponteiro do
                                     ;BASIC
            call    inifcb           ;inicializa o fcb
            ld      a,d              ;a=número do drive
            ld      de,(fcb)         ;coloca na primeira
            ld      (de),a           ;posição do FCB
            ld      hl,filnam        ;transfere a especificação
            inc     de               ;do arquivo para o FCB
            ld      bc,0011          ;bc=tamanho do nome
            ldir                     ;transfere
            ld      de,(dma)         ;ajusta o dma
            ld      c,#1a            ;
            call    bdos             ;
            ld      de,(fcb)         ;de aponta para o FCB
            ld      a,(comfind)      ;
```

```
        ld      c,a             ;c=função #11 do bdos
        call    bdos            ;
        ld      hl,(auxiliar)   ;obtém o ptr. do BASIC
        or      a               ;testa se achou ou não
        jp      nz,volta        ;Não, volta para o BASIC
        ld      de,(dma)        ;Sim, transfere o nome
        push    de              ;
        push    de              ;para a variável string
        pop     hl              ;hl aponta para o fim do nome
        inc     hl              ;de aponta para a ram livre
        ld      bc,#0008        ;obtém a extensão
        ldir                    ;
        ld      a,"."           ;coloca o ponto separando
        ld      (de),a          ;o nome da extensão
        pop     de              ;
        ld      hl,(endvar)     ;
        ld      (hl),#0c        ;ajusta o tamanho da string
        inc     hl
        ld      (hl),e          ;ajusta o endereço da
        inc     hl              ;string propriamente dita
        ld      (hl),d          ;no buffer da variável.
        ld      hl,(auxiliar)   ;obtém o ptr. do BASIC
        jp      volta           ;retorna


;a rotina inifcb promove a inicialização do
;FCB usando a instrução LDIR para preencher
;uma área com um único dado (#00 no caso). A
;inicialização do FCB é uma tarefa fundamental
;na abertura e criação de arquivos.

inifcb
        push    bc              ;salva os registros
        push    hl              ;afetados por esta
        push    de              ;rotina
        ld      hl,(fcb)        ;hl aponta para o FCB
        ld      (hl),#00        ;preenche o primeiro
        push    hl              ;byte com zero
        pop     de              ;de aponta para o 2o.
        inc     de              ;byte do FCB
        ld      bc,0035         ;bc=tamanho do FCB-1
        ldir                    ;preenche todo o FCB
                                ;com zeros
        pop     de              ;recupera os registros
        pop     hl              ;salvos
        pop     bc              ;
        ret                     ;retorna
```

```
;a rotina rdstick lê o estado dos joysticks e das teclas
;do cursor

rdstick
                push    ix              ;salva os registros
                push    iy              ;afetados
                push    bc              ;
                push    de              ;
                push    hl              ;
rdstick_1
                ld      a,#00           ;lê o estado
                call    gtstck          ;das teclas do cursor
                or      a               ;alguma tecla pressionada?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#01           ;lê o estado do
                call    gtstck          ;joystick na porta A
                or      a               ;algum movimento?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#02           ;lê o estado do
                call    gtstck          ;joystick na porta B
                or      a               ;algum movimento?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#00           ;lê o estado da barra
                call    gttrig          ;de espaço
                or      a               ;foi pressionada?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#01           ;lê o estado do botão de
                call    gttrig          ;tiro do joystick A
                or      a               ;foi pressionado?
                jr      nz,fimstick     ;Sim, vai para fimstick
                ld      a,#02           ;lê o estado do botão de
                call    gttrig          ;tiro do joystick B
                or      a               ;foi pressionado?
                jr      z,rdstick_1     ;Não, volta para rdstick_1

fimstick        push    af              ;salva o valor lido
                ld      hl,#8000        ;provoca um pequeno
loopendst       dec     hl              ;retardo para que as
                ld      a,h             ;seleções não sejam
                or      l               ;rápidas demais
                jr      nz,loopendst    ;
                pop     af              ;recupera o valor lido
                pop     hl              ;recupera todos os
                pop     de              ;registros salvos
                pop     bc              ;no início da rotina
```

```
                pop     iy                  ;
                pop     ix                  ;
                ret                         ;retorna


;a rotina calpadvid calcula os padrões do vídeo: a tabela de
;padrões e o número de colunas. Os valores resultantes são
;colocados em tabpad e colunas, respectivamente.

calpadvid
                ex      af,af'              ;salva o registro af
                exx                         ;salva os demais registros
                ld      hl,(txtcgp)         ;obtém a tab. de padrões
                ld      a,(versao)          ;obtém a versão
                or      a                   ;do MSX. É MSX1?
                jr      z,calpad_1          ;Sim, vai para calpad_1
                ld      a,(linlen)          ;obtém o tamanho da tela
                cp      41                  ;O MSX2 está em 80 colunas
                jr      c,calpad_1          ;Não, vai para calpad_1
                ld      a,80                ;Sim, a=80 colunas
                add     hl,hl               ;calcula novo end. da
                                            ;tabela de padrões
                jr      calpad_2            ;vai para calpad_2
calpad_1        ld      a,40                ;a=40 colunas
calpad_2        ld      (colunas),a         ;salva as colunas
                ld      (tabpad),hl         ;salva o end. da tab. de
                                            ;padrões
                exx                         ;recupera os registros salvos
                ex      af,af'              ;
                ret                         ;retorna


;a rotina lefrase transporta da tela para o buffer bufnor
;a sentença a inverter

lefrase
                push    af                  ;salva os registros
                push    bc                  ;modificados por esta
                push    de                  ;rotina
                push    hl                  ;
                call    calendfra           ;calcula o end. da string
                                            ;na VRAM
                call    setvdprd            ;prepara o VDP para leitura
                ld      b,(ix+#00)          ;obtém o comp. da string
                ld      hl,bufnor+#03       ;hl->início da string
                call    lelinha             ;lê a string
                pop     hl                  ;recupera os registros
```

```
                pop     de                  ;salvos
                pop     bc                  ;
                pop     af                  ;
                ret                         ;retorna


;a rotina calendfra, baseada nas coordenadas fornecidas, calcula o endereço da sentença a
;inverter na memória VRAM

calendfra
                ld      h,#00               ;calcula o end. da string
                ld      d,h                 ;na memória VRAM, baseada
                ld      a,(grpacy)          ;nas coordenadas passadas
                ld      l,a                 ;
                or      a                   ;a string está na linha 0?
                jr      z,calend1           ;Sim, vai para calend1
                ld      l,h                 ;Não, calcula o end. do
                ld      b,a                 ;início da linha
                ld      a,(colunas)         ;
                ld      e,a                 ;
loopcal_1       add     hl,de               ;
                djnz    loopcal_1           ;
calend1         ld      a,(grpacx)          ;obtém a coord. x
                ld      e,a                 ;e=coord. x
                add     hl,de               ;soma ao end. já encontrado
                ld      (bufnor+ =#01)      ,hl;salva no buffer das strings
                ld      (bufinv+#01)        ,hl;normal e invertida
                ret                         ;retorna


;a rotina posit posiciona o cursor nas coordenadas passadas
;por hl. h=x e l=y

posit
                push    af                  ;salva todos os
                push    bc                  ;registros afetados
                push    de                  ;por esta rotina
                push    hl                  ;
                ex      de,hl               ;troca o conteúdo de hl pelo
                                            ;de de
                ld      hl,#0000            ;zera hl
                ld      a,e                 ;a=linha
                or      a                   ;é igual a zero?
                jr      z,posit2            ;Sim, vai para posit2
                push    de                  ;Não, salva as coordenadas
                ld      b,a                 ;b=núm. da linha
                ld      a,(colunas)         ;a=núm. de colunas
```

```
                ld      e,a             ;e=núm. de colunas
                ld      d,#00           ;de=núm. de colunas
posit1          add     hl,de           ;hl=hl+núm. de colunas
                djnz    posit1
                pop     de              ;recupera as coordenadas
posit2          ld      a,d             ;a=coluna
                or      a               ;é igual a zero?
                jr      z,posit3        ;Sim, vai para posit3
                ld      e,a             ;Não, e=coluna
                ld      d,#00           ;de=coluna
                add     hl,de           ;hl aponta para o end. da
                                        ;VRAM
                                        ;correspondente a x1,y1
posit3          call    setvdpwt        ;prepara o VDP para escrita
                pop     hl              ;recupera os registros
                pop     de              ;
                pop     bc              ;
                pop     af              ;
                ret                     ;e retorna


;início da lista com os nomes dos novos comandos e
;endereços de chamada

lista
                defm    "REVERSE"       ;nome do comando
                defb    #00             ;fim do nome
                defw    inverte         ;end. da rotina
                defm    "CLRSCR"        ;nome do comando
                defb    #00             ;fim do nome
                defw    newcls          ;end. da rotina
                defm    "WINDOW"        ;nome do comando
                defb    #00             ;fim do nome
                defw    window          ;end. da rotina
                defm    "MENU"          ;nome do comando
                defb    #00             ;fim do nome
                defw    menu            ;end. da rotina
                defm    "FINDFIRST"     ;nome do comando
                defb    #00             ;fim do nome
                defw    findfirst       ;end. da rotina
                defm    "FIND"          ;nome do comando
                defb    #83             ;token do next
                defb    #00             ;fim do nome
                defw    findnext        ;end. da rotina
endlista        defb    #00             ;fim da lista
```

```
;rotinas para o acesso direto à RAM de vídeo

rdvram
            call    setvdprd        ;ajusta o VDP para leitura
            in      a,(#98)         ;lê um byte
            ret                     ;retorna


wtvram
            push    af              ;salva o dado a enviar
            call    setvdpwt        ;ajusta o VDP para escrita
            pop     af              ;recupera o dado a enviar
            out     (#98),a         ;envia
            ret                     ;retorna

setvdprd
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,rdvram1       ;Sim, vai para rdvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a         ;
rdvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde será
            and     #3f             ;lido o dado
            out     (#99),a         ;
            ex      (sp),hl         ;demora para
            ex      (sp),hl         ;sincronização
            ret


setvdpwt
            ld      a,(versao)      ;obtém a versão do MSX
            or      a               ;é MSX1?
            jr      z,wtvram1       ;Sim, vai para wtvram1
            xor     a               ;Não, inicializa o VDP
            out     (#99),a         ;do MSX2
            ld      a,#8e
            out     (#99),a
wtvram1     ld      a,l             ;informa ao
            out     (#99),a         ;VDP o endereço na
            ld      a,h             ;VRAM onde o
            and     #3f             ;dado será
            or      #40             ;gravado
            out     (#99),a         ;
```

```
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                ret                     ;retorna


;a rotina lelinha transporta uma linha da VRAM para a RAM

lelinha
                in      a,(#98)         ;lê o carac. da VRAM
                ld      (hl),a          ;salva-o no buffer apontado
                                        ;por hl
                inc     hl              ;incrementa o ponteiro do
                                        ;buffer
                djnz    lelinha         ;prepara a próxima leitura na
                                        ;tela
                ret                     ;retorna


;a rotina esclinha transporta uma linha da RAM para a VRAM

esclinha
                ld      a,(hl)          ;lê o carac. do buffer
                out     (#98),a         ;escreve-o na VRAM
                inc     hl              ;incrementa o ponteiro do
                                        ;buffer
                djnz    esclinha        ;prepara a próxima escrita na
                                        ;tela
                ret                     ;retorna


;área das variáveis usadas no programa

bufnor
                defb    #00             ;tamanho da string
                defw    #0000           ;posição de escrita
                defs    33              ;string

bufinv
                defb    #00             ;tamanho da string
                defw    #0000           ;posição de escrita
                defs    33              ;string

inversao        defb    #00             ;flag de campo já invertido

compstr         defb    #00             ;comprimento da string

colunas         defb    #00             ;número de colunas na tela
```

```
tabpad          defw    #0000           ;endereço da tab. de padrões

auxiliar        defw    #0000           ;ponteiro do BASIC

endvar          defw    #0000           ;end. da variável BASIC

comfind         defb    #00             ;número da função de procura

opcao           defb    #00             ;valor da opção atual

opcaofim        defb    #00             ;valor da opção final

cooratual                               ;coordenadas atuais
yatual  defb    #00                     ;y
xatual  defb    #00                     ;x

bufferlinha
                defs    81              ;buffer de linha da tela

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa que implementa os comandos FINDFIRST e FINDNEXT:

```
10 FOR A%=&HD000 TO &HD869
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG29.BIN",&HD000,&HD869
100 DATA F3,DB,A8,47,E6,F0,0F,0F,0F,0F,B0,D3,A8,E6,03,32
110 DATA 2E,D0,78,21,2D,D0,11,B1,FF,01,05,00,ED,B0,21,32
120 DATA D0,11,00,40,01,37,08,ED,B0,D3,A8,FB,C9,F7,00,00
130 DATA 40,C9,F5,C5,D5,E5,7B,FE,02,C2,20,40,2A,AF,F6,7E
140 DATA B7,C2,16,40,23,23,23,23,DD,21,66,46,CD,B3,40,CD
150 DATA 25,40,E1,D1,C1,F1,C9,22,DC,47,21,09,47,01,37,00
160 DATA ED,5B,DC,47,1A,ED,B1,E2,48,40,13,1A,BE,C2,2E,40
170 DATA 23,7E,FE,00,CA,49,40,C3,38,40,C9,23,4E,23,46,F1
180 DATA C5,ED,53,DC,47,EB,C9,DD,E5,DD,E5,DD,21,66,46,CD
190 DATA B3,40,DD,E1,28,1A,FE,2C,28,12,2B,DD,E5,C5,DD,21
200 DATA 0E,52,CD,B3,40,C1,DD,E1,7B,DD,77,00,DD,23,10,D9
210 DATA DD,E1,C9,3E,05,32,14,F4,18,07,3E,0D,32,14,F4,18
220 DATA 00,E1,D1,C1,F1,3A,14,F4,5F,C9,2B,AF,32,14,F4,D1
230 DATA D1,C1,F1,D1,C1,F1,DD,E1,DD,E1,F5,C5,D5,DD,21,66
240 DATA 46,C3,B3,40,C9,CD,59,01,C9,06,04,DD,21,9B,41,CD
250 DATA 55,40,22,DC,47,CD,CC,40,2A,DC,47,C3,98,40,DD,7E
```

```
260 DATA 00,32,B7,FC,DD,7E,01,32,B9,FC,DD,7E,02,32,D8,47
270 DATA FE,21,D2,93,41,DD,7E,03,B7,28,1C,3A,D7,47,B7,28
280 DATA 16,3A,8F,47,B7,28,10,2A,90,47,CD,67,47,21,92,47
290 DATA 3A,8F,47,47,CD,88,47,2A,DC,47,3A,D8,47,B7,CA,97
300 DATA 41,DD,21,8F,47,FD,21,B3,47,DD,77,00,FD,77,00,CD
310 DATA 88,46,3A,B7,FC,5F,3A,D9,47,BB,DA,93,41,3A,B9,FC
320 DATA FE,18,D2,93,41,CD,AA,46,F3,2A,DA,47,E5,11,00,07
330 DATA 19,EB,0E,E0,DD,46,00,DD,21,92,47,FD,21,B6,47,E1
340 DATA C5,E5,D5,11,08,00,DD,46,00,19,10,FD,D1,06,08,CD
350 DATA 41,47,2F,EB,CD,47,47,EB,23,13,10,F3,E1,C1,FD,71
360 DATA 00,0C,DD,23,FD,23,10,D8,FD,21,B3,47,FD,6E,01,FD
370 DATA 66,02,CD,67,47,FD,46,00,21,B6,47,CD,88,47,3E,FF
380 DATA 32,D7,47,FB,C9,F1,C3,81,40,F1,C3,98,40,00,00,00
390 DATA 00,06,02,DD,21,5C,42,CD,55,40,22,DC,47,DD,7E,00
400 DATA FE,02,D2,81,40,CD,88,46,3A,D9,47,5F,DD,7E,01,1C
410 DATA BB,D2,81,40,B7,CA,98,40,DD,7E,00,B7,C2,13,42,F3
420 DATA DD,46,01,C5,2A,B3,F3,3A,D9,47,5F,16,00,3A,B1,F3
430 DATA 47,C5,CD,4F,47,3A,D9,47,D5,E5,21,E5,47,47,CD,81
440 DATA 47,3E,20,77,E1,D1,CD,67,47,D5,E5,21,E6,47,3A,D9
450 DATA 47,47,CD,88,47,E1,D1,19,C1,10,D6,C1,10,C5,FB,2A
460 DATA DC,47,C3,98,40,F3,DD,46,01,C5,2A,B3,F3,3A,D9,47
470 DATA 5F,16,00,3A,B1,F3,47,C5,CD,4F,47,3A,D9,47,D5,E5
480 DATA 21,E6,47,47,CD,81,47,3E,20,21,E5,47,77,E1,D1,CD
490 DATA 67,47,D5,E5,21,E5,47,3A,D9,47,47,CD,88,47,E1,D1
500 DATA 19,C1,10,D3,C1,10,C2,FB,2A,DC,47,C3,98,40,00,00
510 DATA DD,21,7D,43,06,05,CD,55,40,22,DC,47,CD,88,46,DD
520 DATA 21,7D,43,DD,66,00,DD,6E,01,DD,56,02,DD,5E,03,7C
530 DATA BA,D2,81,40,7D,BB,D2,81,40,7B,FE,18,D2,81,40,3A
540 DATA D9,47,3D,BA,DA,81,40,DD,7E,04,B7,C2,A5,42,CD,1F
550 DATA 43,2A,DC,47,C3,98,40,AF,32,82,43,32,83,43,22,84
560 DATA 43,ED,53,86,43,7C,82,CB,3F,3D,67,3C,3C,57,7D,83
570 DATA CB,3F,3D,6F,3C,3C,5F,CD,1F,43,CD,E6,42,CD,F5,42
580 DATA CD,1F,43,CD,04,43,CD,11,43,20,EF,2A,84,43,ED,5B
590 DATA 86,43,CD,1F,43,C3,9F,42,3A,85,43,BC,30,03,25,14
600 DATA C9,3E,01,32,82,43,C9,3A,84,43,BD,30,03,2D,1C,C9
610 DATA 3E,01,32,83,43,C9,F5,E5,21,00,10,2B,7C,B5,20,FB
620 DATA E1,F1,C9,3A,82,43,FE,01,28,01,C9,3A,83,43,FE,01
630 DATA C9,C5,D5,E5,CD,E1,46,7A,94,3D,47,C5,3E,18,D3,98
640 DATA E3,E3,3E,17,D3,98,E3,E3,10,F8,3E,19,D3,98,C1,6B
650 DATA CD,E1,46,3E,1A,D3,98,E3,E3,3E,17,D3,98,E3,E3,10
660 DATA F8,3E,1B,D3,98,E1,D1,D5,E5,7B,95,3D,47,7A,94,3D
670 DATA 4F,2C,CD,E1,46,3E,16,D3,98,C5,41,3E,20,D3,98,E3
680 DATA E3,10,F8,3E,16,D3,98,2C,C1,10,E7,E1,D1,C1,C9,00
690 DATA 00,00,00,00,00,00,00,00,00,00,DD,21,83,45,06,06
700 DATA CD,55,40,2B,DD,21,66,46,CD,B3,40,FE,2C,C2,20,40
710 DATA DD,21,66,46,CD,B3,40,CA,20,40,DD,21,A4,5E,CD,B3
720 DATA 40,ED,53,DE,47,3A,63,F6,FE,02,C2,88,40,22,DC,47
```

```
730 DATA DD,21,9B,41,FD,21,83,45,FD,7E,02,FD,BE,00,DA,81
740 DATA 40,FD,7E,03,FD,BE,01,DA,81,40,CD,88,46,3A,D9,47
750 DATA 3D,FD,BE,00,DA,81,40,FD,BE,02,DA,81,40,3E,18,FD
760 DATA BE,01,DA,81,40,FD,BE,03,DA,81,40,FD,66,00,FD,6E
770 DATA 01,FD,56,02,FD,5E,03,FD,7E,04,DD,77,02,3E,01,DD
780 DATA 77,03,22,E3,47,AF,32,E1,47,32,D7,47,7C,BA,CA,D3
790 DATA 44,7D,BB,C2,20,40,3A,87,45,5F,3A,88,45,83,5F,3A
800 DATA 83,45,4F,3A,85,45,06,00,93,38,04,04,B9,20,F9,78
810 DATA 32,E2,47,DD,21,9B,41,2A,E3,47,7C,DD,77,00,7D,DD
820 DATA 77,01,CD,CC,40,CD,3F,46,FE,03,CA,67,44,FE,07,CA
830 DATA 9B,44,FE,FF,CA,70,45,18,DA,3A,88,45,5F,3A,E4,47
840 DATA 83,5F,3A,87,45,83,5F,3A,85,45,BB,38,0E,7B,32,E4
850 DATA 47,3A,E1,47,3C,32,E1,47,C3,41,44,3A,83,45,67,3A
860 DATA 84,45,6F,22,E3,47,AF,32,E1,47,C3,41,44,3A,88,45
870 DATA 5F,3A,E4,47,4F,3A,83,45,B9,28,16,79,93,5F,3A,87
880 DATA 45,4F,7B,91,32,E4,47,3A,E1,47,3D,32,E1,47,C3,41
890 DATA 44,3A,85,45,67,3A,86,45,6F,22,E3,47,3A,E2,47,32
900 DATA E1,47,C3,41,44,3A,88,45,5F,3A,84,45,4F,3A,86,45
910 DATA 06,00,93,04,B9,20,FB,78,32,E2,47,DD,21,9B,41,2A
920 DATA E3,47,7C,DD,77,00,7D,DD,77,01,CD,CC,40,CD,3F,46
930 DATA FE,05,CA,10,45,FE,01,CA,40,45,FE,FF,CA,70,45,C3
940 DATA E9,44,3A,88,45,5F,3A,E3,47,83,5F,3A,86,45,BB,DA
950 DATA 2E,45,7B,32,E3,47,3A,E1,47,3C,32,E1,47,C3,E9,44
960 DATA 3A,83,45,67,3A,84,45,6F,22,E3,47,AF,32,E1,47,C3
970 DATA E9,44,3A,88,45,5F,3A,84,45,4F,3A,E3,47,B9,28,0E
980 DATA 93,32,E3,47,3A,E1,47,3D,32,E1,47,C3,E9,44,3A,85
990 DATA 45,67,3A,86,45,6F,22,E3,47,3A,E2,47,32,E1,47,C3
1000 DATA E9,44,2A,DE,47,3A,E1,47,77,23,36,00,CD,56,01,2A
1010 DATA DC,47,C3,98,40,00,00,00,00,00,00,3E,11,32,E0,47
1020 DATA 18,05,3E,12,32,E0,47,DD,21,66,46,CD,B3,40,CA,20
1030 DATA 40,DA,20,40,DD,21,A4,5E,CD,B3,40,ED,53,DE,47,3A
1040 DATA 63,F6,FE,03,C2,88,40,AF,12,13,12,13,12,2B,DD,21
1050 DATA 66,46,CD,B3,40,FE,2C,C2,20,40,DD,21,66,46,CD,B3
1060 DATA 40,FE,22,C2,20,40,DD,21,0E,6A,CD,B3,40,22,DC,47
1070 DATA CD,2B,46,7A,ED,5B,53,F3,12,21,66,F8,13,01,0B,00
1080 DATA ED,B0,ED,5B,4F,F3,0E,1A,CD,7D,F3,ED,5B,53,F3,3A
1090 DATA E0,47,4F,CD,7D,F3,2A,DC,47,B7,C2,98,40,ED,5B,4F
1100 DATA F3,D5,D5,E1,23,01,08,00,ED,B0,3E,2E,12,D1,2A,DE
1110 DATA 47,36,0C,23,73,23,72,2A,DC,47,C3,98,40,C5,E5,D5
1120 DATA 2A,53,F3,36,00,E5,D1,13,01,23,00,ED,B0,D1,E1,C1
1130 DATA C9,DD,E5,FD,E5,C5,D5,E5,3E,00,CD,D5,00,B7,20,28
1140 DATA 3E,01,CD,D5,00,B7,20,20,3E,02,CD,D5,00,B7,20,18
1150 DATA 3E,00,CD,D8,00,B7,20,10,3E,01,CD,D8,00,B7,20,08
1160 DATA 3E,02,CD,D8,00,B7,28,D0,F5,21,00,80,2B,7C,B5,20
1170 DATA FB,F1,E1,D1,C1,FD,E1,DD,E1,C9,08,D9,2A,B7,F3,3A
1180 DATA 2D,00,B7,28,0C,3A,B0,F3,FE,29,38,05,3E,50,29,18
1190 DATA 02,3E,28,32,D9,47,22,DA,47,D9,08,C9,F5,C5,D5,E5
```

```
1200 DATA CD,C2,46,CD,4F,47,DD,46,00,21,92,47,CD,81,47,E1
1210 DATA D1,C1,F1,C9,26,00,54,3A,B9,FC,6F,B7,28,09,6C,47
1220 DATA 3A,D9,47,5F,19,10,FD,3A,B7,FC,5F,19,22,90,47,22
1230 DATA B4,47,C9,F5,C5,D5,E5,EB,21,00,00,7B,B7,28,0C,D5
1240 DATA 47,3A,D9,47,5F,16,00,19,10,FD,D1,7A,B7,28,04,5F
1250 DATA 16,00,19,CD,67,47,E1,D1,C1,F1,C9,52,45,56,45,52
1260 DATA 53,45,00,B7,40,43,4C,52,53,43,52,00,9F,41,57,49
1270 DATA 4E,44,4F,57,00,5E,42,4D,45,4E,55,00,88,43,46,49
1280 DATA 4E,44,46,49,52,53,54,00,89,45,46,49,4E,44,83,00
1290 DATA 90,45,00,CD,4F,47,DB,98,C9,F5,CD,67,47,F1,D3,98
1300 DATA C9,3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3
1310 DATA 99,7C,E6,3F,D3,99,E3,E3,C9,3A,2D,00,B7,28,07,AF
1320 DATA D3,99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6,40,D3,99
1330 DATA E3,E3,C9,DB,98,77,23,10,FA,C9,7E,D3,98,23,10,FA
1340 DATA C9,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1350 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1360 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1370 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1380 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1390 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1400 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1410 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1420 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1430 DATA 00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
1440 DATA 00,00,00,00,00,00,00,00,FF,FF
```

Listagem do programa de teste:

```
100 CLS:KEYOFF:WIDTH 39
110 BLOAD"PROG29.BIN",R
120 REM *** DEMONSTRAÇÃO DOS COMANDOS ***
130 REM *** FINDFIRST E FINDNEXT ***
140 FINDFIRST A$,"????????.???"
150 REM *** A$=PRIMEIRO ARQUIVO ***
160 IF A$="" THEN GOTO 220:REM *** SEM ARQUIVOS ***
170 REM *** IMPRIME O DIRETÓRIO ***
180 S1$="DIRETÓRIO DO DRIVE A"
190 X=(40-LEN(S1$))/2:Y=6:LOCATE X,Y:PRINT S1$:PRINT:PRINT A$;" ";
200 FINDNEXT A$,"????????.???":IF A$<>"" THEN PRINT A$;" ";:GOTO 200
210 END
220 REM *** EMITE MENSAGEM DE DISCO ***
230 REM *** SEM ARQUIVOS ***
240 LOCATE 0,10:PRINT "O DISQUETE NÃO CONTÉM ARQUIVOS"
250 END
```

COMENTÁRIOS SOBRE O PROGRAMA QUE IMPLEMENTA OS COMANDOS FINDFIRST E FINDNEXT

O programa acima faz uso de alguns endereços do BASIC de Disco que já vimos anteriormente. Esses endereços não fornecem os vetores para a área do FCB e para o DMA do BASIC de disco. Você deve estar lembrado que, na função #11, eu afirmei que o FCB do primeiro arquivo que se correspondesse à especificação colocada no FCB seria colocado no DMA. Assim sendo, precisávamos fazer uso dos ponteiros para o FCB (contendo a descrição do arquivo a procurar) e para o DMA (contendo o FCB do arquivo encontrado). A rotina é muito simples de se entender, bastando uma análise dos comentários feitos após cada instrução. A única chamada que não foi comentada refere-se à rotina **ANFILE** (nome dado por mim), que faz a análise de uma especificação de arquivo fornecida pelo BASIC. Ao chamar esta rotina do interpretador BASIC, devemos fazer com que HL aponte para o primeiro par de aspas (") da especificação do arquivo. Esta rotina transfere o nome (ou especificação) do arquivo para a variável do sistema chamada **FILNAM**, já na forma a ser usada pelo FCB. No retorno, o registro D contém o número do drive a ser acessado. Quer mais "moleza" que isto? O que se segue são simples arrumações, para que na volta ao BASIC o nome do arquivo encontrado seja transferido para a variável correta. Examine a listagem do programa de teste para entender a sintaxe dos novos comandos.

O PROGRAMA QUE APRESENTA AS CARACTERÍSTICAS DE UM DISQUETE

Gostaria, agora, de apresentar um pequeno programa que apresenta todas as informações contidas no BPB de um determinado drive. Como condição, devemos usar somente rotinas da BDOS (até para a impressão no vídeo). Vamos então às listagens.

Listagem em assembly Z-80 do código-fonte do programa que apresenta as informações no BPB:

```
bufset  equ     #f351
bdos    equ     #f37d
valtyp  equ     #f663
argusr  equ     #f7f8

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

                org     #d100
```

```
inicio
          ld      a,(valtyp)       ;obtém o tipo de argumento
          cp      #02              ;se o argumento da
                                   ;função USR não for
                                   ;inteiro
          ret     nz               ;retorna
          ld      a,(argusr)       ;drive > 2 (B)?
          cp      #03              ;
          jr      nc,erro          ;Sim, volta com erro
          or      a                ;igual a zero?
          jr      z,erro           ;Sim, volta com erro
          ld      e,a              ;inicializa o BPB
          ld      c,#1b            ;do drive em
          call    bdos             ;questão
          cp      #ff              ;houve erro?
          jr      z,erro           ;Sim, vai para erro
inicio_1  push    ix               ;salva o ponteiro
                                   ;para o BPB
          ld      de,mensagem_1    ;imprime a primeira
          ld      c,#09            ;mensagem
          call    bdos             ;
          pop     ix               ;recupera o ptr.
          push    ix               ;torna a salvar
          ld      l,(ix+#0e)       ;hl=número de
          ld      h,(ix+#0f)       ;aglomerados
          call    ascii            ;acha os dígitos
          ld      de,mensagem_2    ;imprime a segunda
          ld      c,#09            ;mensagem
          call    bdos             ;
          pop     ix               ;recupera o ptr.
          push    ix               ;torna a salvar
          ld      l,(ix+#11)       ;hl=número do
          ld      h,(ix+#12)       ;setor inicial
                                   ;do diretório
          call    ascii            ;imprime os dígitos
          ld      de,mensagem_3    ;imprime a terceira
          ld      c,#09            ;mensagem
          call    bdos             ;
          pop     ix               ;recupera o ptr.
          ld      l,(ix+#0c)       ;hl=setor inicial
          ld      h,(ix+#0d)       ;da área de dados
          push    hl               ;salva
          dec     hl               ;hl=setor final do
                                   ;diretório
          call    ascii            ;imprime os dígitos
          ld      de,mensagem_4    ;imprime a quarta
          ld      c,#09            ;mensagem
```

```
          call    bdos              ;
          pop     hl                ;recupera hl
          call    ascii             ;imprime os dígitos
          ld      hl,#0000          ;sinaliza que correu
          ld      (argusr),hl       ;tudo bem e
          ret                       ;volta ao BASIC


erro
          ld      hl,#ffff          ;sinaliza a ocorrência
          ld      (argusr),hl       ;de um erro
          ret                       ;e volta ao BASIC


;a rotina ascii converte um valor de 16 bits
;para os caracteres ASCII correspondentes. Na
;entrada, o par HL deverá conter o número a
;converter e imprimir

ascii
          push    af                ;salva os registros
          push    bc                ;alterados por
          push    de                ;esta rotina
          push    ix                ;
          push    ix                ;
          push    iy                ;
          ld      ix,valorasc       ;ix aponta p/ valorasc
          ld      iy,dezmil         ;ix aponta p/ dezmil
          ld      e,(iy+#00)        ;de=10.000
          ld      d,(iy+#01)        ;
          xor     a                 ;zera a flag de carry
          sbc     hl,de             ;subtrai para testar
                                    ;se o número >= 10.000
          jr      nc,ascii1         ;Se for maior, vai para ascii1
          add     hl,de             ;Se não, soma os 10.000
          inc     iy                ;incrementa o ponteiro
          inc     iy                ;
          ld      e,(iy+#00)        ;de=1.000
          ld      d,(iy+#01)        ;
          xor     a                 ;zera a flag de carry
          sbc     hl,de             ;número >= 1.000 ?
          jr      nc,ascii1         ;Sim, vai para ascii1
          add     hl,de             ;Não, repõe os 1.000
          inc     iy                ;incrementa o ponteiro
          inc     iy                ;
          ld      e,(iy+#00)        ;de=100
          ld      d,(iy+#01)        ;
```

```
                xor     a               ;zera a flag de carry
                sbc     hl,de           ;número >= 100 ?
                jr      nc,ascii1       ;Sim, vai para ascii1
                add     hl,de           ;Não, repõe os 100
                inc     iy              ;incrementa o ponteiro
                inc     iy              ;
                ld      e,(iy+#00)      ;de=10
                ld      d,(iy+#01)      ;
                xor     a               ;zera a flag de carry
                sbc     hl,de           ;número >= 10 ?
                jr      nc,ascii1       ;Sim, vai para ascii1
                inc     iy              ;Não, incrementa o
                inc     iy              ;ponteiro para as
                                        ;unidades
ascii1          add     hl,de           ;repõe o valor tirado
ascii2          xor     a               ;zera a flag de carry
                                        ;e o contador
                ld      e,(iy+#00)      ;de=quantidade a
                ld      d,(iy+#01)      ;subtrair
ascii3          sbc     hl,de           ;subtrai
                inc     a               ;incrementa o contador
                jr      nc,ascii3       ;ix<de? Não, continua
                dec     a               ;Sim, decrementa o
                                        ;contador
                add     hl,de           ;repõe o valor tirado
                                        ;em excesso
                add     a,#30           ;soma #30 ao contador
                                        ;para obter o ASCII
                                        ;correspondente
                ld      (ix+#00),a      ;salva no buffer
                inc     ix              ;incrementa o ptr. ix
                dec     e               ;chegou às unidades?
                jr      z,volta         ;Sim, sai
                inc     iy              ;Não, obtém o próximo
                inc     iy              ;dígito
                jr      ascii2          ;
volta           ld      a,"$"           ;a=símbolo de término
                                        ;de string usado pelo
                                        ;BDOS
                ld      (ix+#00),a      ;salva
                ld      de,valorasc     ;imprime os
                ld      c,#09           ;dígitos
                call    bdos            ;
                call    linefeed        ;avança a linha
                pop     iy              ;recupera os
                pop     ix              ;registros salvos
                pop     hl              ;
```

```
                pop     de                      ;
                pop     bc                      ;
                pop     af                      ;
                ret                             ;e retorna


;tabascii é uma tabela usada pela rotina ascii.

tabascii

dezmil  defw    10000
mil     defw    1000
cent    defw    100
dez     defw    10
uni     defw    1

valorasc defs   10                      ;este buffer àrmazena
                                        ;os dígitos em ASCII


;a rotina linefeed avança a linha

linefeed
                ld      e,#0d           ;e=carriage return
                ld      c,#02           ;envia para o vídeo
                call    bdos            ;
                ld      e,#0a           ;e=line feed
                ld      c,#02           ;envia para o vídeo
                call    bdos            ;
                ret                     ;e retorna


mensagem_1      defm    "Núm. total de aglomerados      : $"
mensagem_2      defm    "Setor inicial do diretório     : $"
mensagem_3      defm    "Setor final do diretório       : $"
mensagem_4      defm    "Setor inicial da área de dados : $"

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa que apresenta as informações no BPB:

```
10 FOR A%=&HD100 TO &HD2A8
20 READ B$
30 POKE A%,VAL("&H"+B$)
```

```
40 NEXT A%
50 BSAVE "PROG30.BIN",&HD100,&HD2A8
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,FE,03,30,61,B7,28,5E
110 DATA 5F,0E,1B,CD,7D,F3,FE,FF,28,54,DD,E5,11,20,D2,0E
120 DATA 09,CD,7D,F3,DD,E1,DD,E5,DD,6E,0E,DD,66,0F,CD,75
130 DATA D1,11,42,D2,0E,09,CD,7D,F3,DD,E1,DD,E5,DD,6E,11
140 DATA DD,66,12,CD,75,D1,11,64,D2,0E,09,CD,7D,F3,DD,E1
150 DATA DD,6E,0C,DD,66,0D,E5,2B,CD,75,D1,11,86,D2,0E,09
160 DATA CD,7D,F3,E1,CD,75,D1,21,00,00,22,F8,F7,C9,21,FF
170 DATA FF,22,F8,F7,C9,F5,C5,D5,DD,E5,DD,E5,FD,E5,DD,21
180 DATA 07,D2,FD,21,FD,D1,FD,5E,00,FD,56,01,AF,ED,52,30
190 DATA 34,19,FD,23,FD,23,FD,5E,00,FD,56,01,AF,ED,52,30
200 DATA 24,19,FD,23,FD,23,FD,5E,00,FD,56,01,AF,ED,52,30
210 DATA 14,19,FD,23,FD,23,FD,5E,00,FD,56,01,AF,ED,52,30
220 DATA 04,FD,23,FD,23,19,AF,FD,5E,00,FD,56,01,ED,52,3C
230 DATA 30,FB,3D,19,C6,30,DD,77,00,DD,23,1D,28,06,FD,23
240 DATA FD,23,18,E2,3E,24,DD,77,00,11,07,D2,0E,09,CD,7D
250 DATA F3,CD,11,D2,FD,E1,DD,E1,E1,D1,C1,F1,C9,10,27,E8
260 DATA 03,64,00,0A,00,01,00,00,00,00,00,00,00,00,00,00
270 DATA 00,1E,0D,0E,02,CD,7D,F3,1E,0A,0E,02,CD,7D,F3,C9
280 DATA 4E,A3,6D,2E,20,74,6F,74,61,6C,20,64,65,20,61,67
290 DATA 6C,6F,6D,65,72,61,64,6F,73,20,20,20,20,20,20,3A
300 DATA 20,24,53,65,74,6F,72,20,69,6E,69,63,69,61,6C,20
310 DATA 64,6F,20,64,69,72,65,74,A2,72,69,6F,20,20,20,20
320 DATA 20,3A,20,24,53,65,74,6F,72,20,66,69,6E,61,6C,20
330 DATA 64,6F,20,64,69,72,65,74,A2,72,69,6F,20,20,20,20
340 DATA 20,20,20,3A,20,24,53,65,74,6F,72,20,69,6E,69,63
350 DATA 69,61,6C,20,64,61,20,A0,72,65,61,20,64,65,20,64
360 DATA 61,64,6F,73,20,3A,20,24,00
```

Listagem do programa de teste:

```
100 CLS:KEYOFF:WIDTH 40
110 BLOAD"PROG30.BIN"
120 DEFUSR=&HD100
130 REM *** IMPRIME O TÍTULO ***
140 S1$="CARACTERÍSTICAS DO DRIVE A"
150 LOCATE (40-LEN(S1$))/2,6:PRINT S1$:PRINT
160 A%=USR(1)
170 IF A%=-1 THEN GOTO 190
180 END
190 REM *** ESPECIFICAÇÃO INVÁLIDA ***
200 PRINT "ESPECIFICAÇÃO INVÁLIDA"
210 END
```

COMENTÁRIOS SOBRE O PROGRAMA QUE APRESENTA AS INFORMAÇÕES DO BPB

O programa acima é uma boa ilustração de algumas funções do BDOS para a parte de impressão no vídeo. A única rotina que merece algum destaque é a que converte um número de até 16 bits contido no par HL em caracteres ASCII equivalentes. Esta rotina se torna absolutamente indispensável quando desejamos apresentar dados numéricos. Apesar de aparentemente complexa, o seu funcionamento pode ser reduzido aos seguintes passos:

1.Compara se o valor enviado por HL é maior ou igual a 10.000. Se for, faz decrementos sucessivos de 10.000 à medida em que incrementa o contador. Assim que o último decremento resultar num valor menor do que 10.000, sai do loop de decremento por este valor; decrementa o contador que, por sua vez, passa a indicar as dezenas de milhares, e imprime o dígito das dezenas de milhares. Se, por exemplo, o valor entrado for 50.000, na saída do loop o registro A (contador) conterá o valor 5, que somado a #30 (ASCII do número 0) resultará no código ASCII do número 5 (#35).

2.Repetir o mesmo processo para os milhares, centenas, dezenas e unidades.

3.Imprimir a string com os dígitos da quantidade numérica já convertida em caracteres ASCII.

Algumas das cartas que recebi me pediam dicas de como acessar a linha de comando no MSX-DOS. O termo *linha de comando* se refere aos comandos passados após o nome de um arquivo. Por exemplo, se você entrasse com o comando

conv a:prog31.gen b:prog31b.gen

no MSX-DOS (após o A>), a linha de comando seria:

a:prog31.gen b:prog31b.gen

incluindo o espaço na frente do **a:**. Vamos, então, passar para um tópico que apresentará as dicas necessárias.

UM POUCO MAIS DO AMBIENTE MSX-DOS

Você já sabe que, no ambiente MSX-DOS, só podemos executar dois tipos de programas: os programas com extensão .BAT (arquivos de lote) e os programas com extensão .COM. Existe ainda um terceiro tipo, o .SYS, que não pode ser acessado através do ambiente MSX-DOS. Este último tipo só pode ser acessado pela rotina de boot (aquela no setor 0 do disquete), já que se trata de um programa que inicializa todo o ambiente desse sistema. Em todo caso, você também pode, se desejar, carregá-lo ao sair de um programa .COM que tenha feito es-

tragos razoáveis na organização do MSX-DOS. Apesar da extensão diferente (.SYS), este arquivo possui exatamente a mesma estrutura de um arquivo .COM, ou seja, é carregado a partir do endereço #0100, e a sua primeira instrução se encontra também nesse endereço. A Tabela 6.7 mostra as principais características dos arquivos .COM e .SYS.

	SYS	COM	BOOT
End. de Carregamento	#0100	#0100	#C000
End. de Execução	#0100	#0100	#C000
Modo de Carregamento	BOOT	MSX-DOS	INTERFACE

Tabela 6.7: Estrutura dos arquivos .SYS e .COM

O termo BOOT na tabela acima se refere ao fato de que os arquivos .SYS podem ser carregados e executados pelas rotinas de boot: a rotina no setor 0 do disquete, acionada pelo reset do microcomputador, e a rotina acionada pela instrução **JP #0000** (em assembly, é claro) a partir de um programa .COM. Este último procedimento só carregará o arquivo .SYS se os estragos feitos na memória pelo programa .COM forem muito grandes.

Vamos passar, agora, a um exemplo que demonstra a utilização da linha de comando nos programas .COM.

UM EXEMPLO DE PROGRAMA NO AMBIENTE MSX-DOS

Para ilustrar mais algumas funções da BDOS e mostrar como interpretar a linha de comando do MSX-DOS, elaborei um pequeno programa de demonstração que tem como finalidade converter o conteúdo de um arquivo ASCII de maiúsculas para minúsculas. A rotina de conversão é feita byte a byte, de modo que este programa é extremamente lento (fica a seu critério aprimorá-lo). Entretanto, se você quiser acelerá-lo, use um buffer para leitura e outro para gravação, com uns 2 Kbytes cada. O algoritmo usado é o seguinte:

1.Inicializar os dois FCBs usados: um para a leitura do arquivo e outro para a gravação do arquivo já convertido.

2.Interpretar o nome do arquivo fornecido na linha de comando. Esta função é realizada pela rotina **lelinha.** Para entender como interpretar a linha de comando, estude bem esta rotina.

3.Tentar abrir o arquivo passado pela linha de comando para testar a sua existência.

4.Se o arquivo existir, renomeá-lo com a extensão .BAK. Desta forma, preserva-se o conteúdo do arquivo original.

5.Criar um arquivo com o nome passado pel.. linha de comando. Este arquivo conterá o conteúdo do arquivo original (agora renomeado como .BAK).

6.Proceder à conversão de maiúsculas para minúsculas, mantendo inalterados todos os textos entre aspas.

7.Fechar o novo arquivo, para que as suas informações sejam atualizadas no diretório. Se não fizermos isto, o novo arquivo apresentará um tamanho de 0 bytes no diretório.

Vamos então à listagem.

Listagem em assembly Z-80 do código-fonte do programa de demonstração:

```
;programa de demonstração
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG31.GEN
;
;onde PROG31.GEN é o nome do arquivo-texto com esta
;listagem

bdos        equu    #0005
dma         equ     #0080


            org     #0100

            call    inifcb              ;inicializa o FCB
            call    lelinha             ;lê o nome do arquivo
            call    copyfcb             ;copia o FCB
            ld      de,fcb1             ;tenta abrir o arquivo
            ld      c,#0f               ;
            call    bdos                ;
            cp      #ff                 ;Houve erro?
            jp      z,erroesp           ;Sim, vai para erroesp
            call    renomeia            ;Não, renomeia
            call    abrearq             ;abre os arquivos
            ld      de,dma              ;ajusta o dma
            ld      c,#1a               ;
            call    bdos                ;
            ld      hl,#0001            ;HL=tamanho do registro
                                        ;no caso, 1 byte
            ld      (fcb1+#0e),hl       ;especifica o tamanho
            ld      (fcb2+#0e),hl       ;
            dec     hl                  ;zera hl
```

```
                ld      (fcb1+#20),hl           ;zera o núm. do registro
                ld      (fcb1+#22),hl           ;inicial
                ld      (fcb1+#24),hl           ;
                ld      (fcb2+#20),hl           ;
                ld      (fcb2+#22),hl           ;
                ld      (fcb2+#24),hl           ;

converte
                call    leregistro              ;lê um registro
                cp      #02                     ;houve erro?
                jp      nc,errolei              ;Sim, vai para errolei
                cp      #01                     ;terminou a leitura
                jr      z,fimconv               ;Sim, termina a conversão
                ld      a,(dma)                 ;obtém o registro lido
                cp      #22                     ;são aspas?
                jr      z,aspas                 ;Sim, vai para aspas
                cp      "A"                     ;as próximas linhas
                                                ;testam
                jr      c,conver_1              ;se o caractere lido é
                cp      #5b                     ;minúsculo ou não
                jr      nc,conver_1             ;Se não for, vai para
                                                ;conver_1
                add     a,#20                   ;Se for, converte
                ld      (dma),a                 ;salva o valor
                                                ;convertido
conver_1        call    gvregistro              ;grava o registro
                cp      #01                     ;disco cheio?
                jp      z,espdisco              ;Sim, vai para espdisco
                or      a                       ;houve erro?
                jp      nz,errogrv              ;Sim, vai para errogrv
                jr      converte                ;Não, faz a próxima
                                                ;leitura

fimconv
                ld      de,fcb2                 ;fecha o arquivo
                ld      c,#10                   ;
                call    bdos                    ;
                jp      #0000                   ;retorna ao sistema


;a rotina aspas espera a ocorrência das próximas aspas
;para continuar a conversão em minúsculas.
;Desta forma, o texto entre aspas não é convertido

aspas
                call    gvregistro              ;salva as aspas
```

```
aspas_1
                call    leregistro          ;lê um registro
                cp      #02                 ;houve erro?
                jp      nc,errolei          ;Sim, vai para errolei
                cp      #01                 ;terminou a leitura
                jr      z,fimconv           ;Sim, termina a
                                            ;conversao
aspas_0         ld      a,(dma)             ;obtém o registro lido
                cp      #22                 ;são aspas?
                jr      z,conver_1          ;Sim, retorna para o
                                            ;loop
                                            ;de conversão
                call    gvregistro          ;Não, grava o registro
                cp      #01                 ;disco cheio?
                jp      z,espdisco          ;Sim, vai para espdisco
                or      a                   ;houve erro?
                jp      nz,errogrv          ;Sim, vai para errogrv
                jr      aspas_1             ;repete a busca das
                                            ;próximas aspas


;a rotina leregistro lê um registro de 1 byte e coloca-o
;no DMA

leregistro
                ld      hl,#0001            ;hl=núm. de registros a
                                            ;ler
                ld      de,fcb1             ;de aponta para o FCB
                ld      c,#27               ;faz a leitura do
                                            ;registro
                call    bdos                ;
                ret                         ;e retorna


;a rotina gvregistro grava um registro de 1 byte

gvregistro
                ld      hl,#0001            ;hl=núm. de registros a
                                            ;gravar
                ld      de,fcb2             ;de=aponta para o FCB
                ld      c,#26               ;grava o registro
                call    bdos                ;
                ret                         ;e retorna
```

```
;a rotina erroesp imprime a mensagem de especificação
;inválida e retorna ao MSX-DOS

erroesp
            ld      de,mensagem_1           ;imprime a mensagem de
            jp      impmens                 ;erro de especificação


;a rotina errolei imprime a mensagem de erro de leitura
;e retorna ao MSX-DOS

errolei
            ld      de,mensagem_2           ;imprime a mensagem de
            jp      impmens                 ;erro de leitura


;a rotina errogrv imprime a mensagem de erro de gravação
;e retorna ao MSX-DOS

errogrv
            ld      de,mensagem_3           ;imprime a mensagem de
            jp      impmens                 ;erro de gravação


;a rotina espdir imprime a mensagem de diretório sem espaço

espdir
            ld      hl,mensagem_4           ;imprime a mensagem
            jp      impmens                 ;


;a rotina espdisco imprime a mensagem de disco sem espaço

espdisco
            ld      hl,mensagem_5           ;imprime a mensagem
            jp      impmens                 ;


;a rotina impmens imprime uma mensagem no video e retorna
;ao MSX-DOS

impmens
            ld      c,#09                   ;imprime a
            call    bdos                    ;mensagem
            jp      #0000                   ;volta ao MSX-DOS
                                            ;com "boot a quente"
```

```
;a rotina renomeia, muda o nome do arquivo original para .BAK

renomeia
        ld      hl,fcb1+#01         ;transfere o nome
        ld      de,fcb1+#11         ;
        ld      bc,#0008            ;
        ldir
        ex      de,hl               ;acrescenta a
        ld      (hl),"B"            ;extensão .BAK
        inc     hl                  ;
        ld      (hl),"A"            ;
        inc     hl                  ;
        ld      (hl),"K"            ;
        ld      de,fcb1             ;faz a renomeação
        ld      c,#17               ;
        call    bdos                ;
        or      a                   ;houve erro?
        jp      nz,errogrv          ;Sim, vai para errogrv
        ld      hl,fcb1+#11         ;Não, transfere o novo
        ld      de,fcb1+#01         ;nome
        ld      bc,#000b            ;
        ldir                        ;
        ld      hl,fcb1+12          ;inicializa o fcb1
        ld      (hl),#00            ;
        ld      de,fcb1+13          ;
        ld      bc,36-12            ;
        ldir                        ;
        ret                         ;retorna


;a rotina abrearq abre dois arquivos:o arquivo original
;para leitura e o arquivo a ser gravado

abrearq
        ld      de,fcb1             ;abre o primeiro arquivo
        ld      c,#0f               ;para leitura
        call    bdos                ;
        ld      de,fcb2             ;abre o segundo
        ld      c,#16               ;arquivo para
        call    bdos                ;gravação
        or      a                   ;houve erro?
        jp      nz,espdir           ;
        ret
```

```
;a rotina lelinha lê a linha de comando passada após o
;nome do programa

lelinha
        ld      de,fcb1             ;de aponta para o FCB
        xor     a                   ;zera o acumulador
        ld      (de),a              ;inicializa o FCB com o
                                    ;drive default
        ld      hl,dma+#01          ;hl aponta para o
                                    ;endereço
                                    ;inicial da linha de
                                    ;comando
        call    pulaesp             ;ignora os espaços e
                                    ;tabulaçoes
        call    testacar            ;obtém o primeiro
                                    ;caractere
                                    ;da especificação
        ld      c,a                 ;salva o caractere em c
        inc     hl                  ;aponta para o próximo
                                    ;caractere
        ld      a,(hl)              ;obtém o caractere
        dec     hl                  ;retorna à posição
                                    ;original
        cp      ":"                 ;é ":" ?
        ld      a,c                 ;recupera o suposto nome
                                    ;do drive
        jr      nz,lenome_ext       ;não é ":". Deve ser o
                                    ;nome do arquivo
                                    ;sem especificação
                                    ;do drive
        inc     hl                  ;Se for ":", incrementa
                                    ;hl
        inc     hl                  ;para apontar o
                                    ;primeiro
                                    ;caractere do nome do
                                    ;arquivo
        sub     #41                 ;subtrai #41 para que o
                                    ;número do drive
                                    ;fique entre 1=A e
                                    ;2=B
        jr      c,espinval          ;Se der carry, a
                                    ;especificação
                                    ;está errada.
        inc     a                   ;a=1=drive A,=2=drive B
        ld      (de),a              ;coloca no 1o. byte do
                                    ;FCB
        jr      lenome_ext          ;lê o nome e a extensão
                                    ;do arquivo
```

```
;a rotina espinval força um erro na abertura do arquivo
;caso a especificaçao do drive esteja errada

espinval
              ld      a,#ff                   ;a=número de drive inválido
              ld      (de),a                  ;coloca no 1o. byte do
                                              ;FCB

;a rotina lenome_ext faz a leitura do nome de um arquivo na
;linha de comando

lenome_ext
              inc     de                      ;incrementa o ponteiro
                                              ;do FCB
              ld      c,#00                   ;registro usado para
                                              ;assinalar a
                                              ;presença de caracteres
                                              ;coringas (* e ?) no
                                              ;nome.
              ld      b,#08                   ;b=núm. de carac. do nome
              call    lenome                  ;lê o nome do arquivo
              ld      a,(hl)                  ;o próximo carac. é "."?
              cp      "."                     ;
              jr      nz,fimnome_ext          ;Não, então acabou a
                                              ;leitura
              inc     hl                      ;Sim, então lê a
                                              ;extensão do
                                              ;arquivo
              ld      b,#03                   ;b=núm. de carac. da
                                              ;extensão
              call    lenome_0                ;lê a extensão
fimnome_ext
              ld      a,c                     ;obtém o sinalizador de
                                              ;caracteres
                                              ;coringas
              ret                             ;e retorna

;a rotina lenome faz a leitura de um determinado número de
;caracteres passados pelo registro B

lenome
```

```
                call    testacar        ;testa o caractere
                jr      c,codesp        ;se for código especial,
                jr      z,codesp        ;vai para codesp
lenome_0        call    testacar        ;testa o caractere
                jr      c,tstfimle      ;chegou ao fim da
                jr      z,tstfimle      ;leitura?
                inc     hl              ;Não, obtém o próximo
                                        ;carac.
                inc     b               ;se b=0, chegou ao fim da
                dec     b               ;leitura
                jr      z,lenome_0      ;obtém o próximo carac. e
                                        ;termina a leitura
                cp      "*"             ;o caractere é *?
                jr      z,coringa       ;Sim, substitui por
                                        ;pontos de
                                        ;interrogação (coringa)
                ld      (de),a          ;Não, coloca o carac. no
                                        ;FCB
                inc     de              ;incrementa o ptr.do FCB
                dec     b               ;decrementa o contador
                                        ;de caracteres a ler
                cp      "?"             ;o último carac. foi "?"?
                jr      z,acheicor      ;Sim, assinala que achou
                                        ;um caractere coringa
                jr      lenome_0        ;fecha o loop

;a rotina coringa faz a substituição do caractere "*"
;pelo número apropriado de "?"

coringa
                call    subscor         ;substitui por coringa
                                        ;(?)
acheicor
                ld      c,#01           ;assinala a presença de
                                        ;caracteres coringas no
                                        ;nome
                                        ;do arquivo

;a rotina codesp incrementa o ponteiro do FCB, no caso
;da especificação do nome conter caracteres especiais,
;forçando assim um erro de especificação

codesp
                ld      a,e             ;a=LSB do ponteiro do
                                        ;FCB
```

```
                add     a,b                 ;soma com o número de
                                            ;caracteres
                                            ;que ainda resta por
                                            ;ler
                ld      e,a                 ;coloca o resultado no
                                            ;LSB
                                            ;do ponteiro do FCB
                ret     nc                  ;volta, se não houve
                                            ;estouro
                inc     d                   ;se houve, ajusta o MSB
                                            ;do ponteiro do FCB
                ret                         ;e retorna


;a rotina tstfimle testa o fim da leitura na eventualidade
;de terem sido entrados menos caracteres (menos de 8 para o
;nome, ou menos de 3 para a extensão) que os permitidos

tstfimle
                inc     b                   ;testa se o contador de
                                            ;caracteres
                dec     b                   ;a ler chegou a zero
                                            ;(acabou a leitura)
                ret     z                   ;Se chegou, volta.
                ld      a,#20               ;Se não, preenche com
                                            ;espaços as
                                            ;posições
                                            ;restantes do nome no
                                            ;FCB
                                            ;
                jr      preenche            ;
subscor
                ld      a,"?"               ;prepara a substituição
                                            ;por coringas (?)
                                            ;das
                                            ;posições restantes do
                                            ;nome
                                            ;no FCB
preenche
                ld      (de),a              ;faz o preenchimento
                inc     de                  ;de acordo com o número
                djnz    preenche            ;de caracteres que ainda
                                            ;faltava
                ret                         ;e retorna
```

```
;a rotina pulaesp pula os espaços e tabulações

pulaesp
            ld      a,(hl)          ;obtém o caractere
            inc     hl              ;incrementa o ponteiro
            call    testaesp        ;testa o caractere
            jr      z,pulaesp       ;se for espaço, testa o
                                    ;caractere seguinte
            dec     hl              ;decrementa o ponteiro
                                    ;para a posição correta
            ret                     ;retorna


;a rotina testacar testa o caractere apontado por hl. Se
;for letra (A-Z ou a-z) ou número (0 a 9), volta com
;NZ e NC, caso contrário, volta com Z,C, ou qualquer combi-
;nação destas duas flags

testacar
            ld      a,(hl)          ;a=caractere
            cp      "a"             ;as próximas 4 linhas
                                    ;testam
            jr      c,testacar_1    ;se o caractere está em
            cp      #7b             ;minúscula. Se não
                                    ;estiver,
            jr      nc,testacar_1   ;pula para testacar_1.
            sub     #20             ;Se estiver, converte
                                    ;em maiúscula

testacar_1
            cp      ":"             ;é ":"?
            ret     z               ;Sim, retorna
            cp      "."             ;é "."?
            ret     z               ;Sim, retorna
            cp      #22             ;é aspas?
            ret     z               ;Sim, retorna
            cp      "["             ;é "["?
            ret     z               ;Sim, retorna
            cp      "]"             ;é "]"?
            ret     z               ;Sim, retorna
            cp      "_"             ;é "_"?
            ret     z               ;Sim, retorna
            cp      "/"             ;é "/"?
            ret     z               ;Sim, retorna
            cp      "+"             ;é "+"?
            ret     z               ;Sim, retorna
            cp      "="             ;é "="?
```

```
          ret     z                        ;Sim, retorna
          cp      ";"                      ;é ";"?
          ret     z                        ;Sim, retorna
          cp      "."                      ;'e ""?
          ret     z                        ;Sim, retorna

testaesp
          cp      #09                      ;é tabulação?
          ret     z                        ;Sim, retorna
          cp      " "                      ;é espaço?
          ret                              ;Retorna


;a rotina inifcb inicializa o FCB. O nome do arquivo é
;inicializado com espaços. Desta forma, se o usuário
;não entrar com a extensão, não ocorrerá nenhum erro.
;O byte do drive (posição #00 do FCB) é inicializado
;com #00 (drive default), assim como todos os registros
;do FCB


inifcb
          ex      af,af'                   ;salva o registro A e
                                           ;as flags
          exx                              ;salva os registros BC,
                                           ;DE e HL
          ld      hl,fcb1                  ;inicializa o FCB com
                                           ;#00
          ld      (hl),#00                 ;
          ld      de,fcb1+#01              ;
          ld      bc,#0023                 ;bc=tamanho do FCB-1
          ldir                             ;preenche o FCB com#00
          ld      hl,fcb1+#01              ;inicializa com espaços o
                                           ;nome do
          ld      (hl),#20                 ;arquivo no FCB
                                           ;
          ld      de,fcb1+#02              ;
          ld      bc,#000a                 ;bc=10=tamanho do nome
                                           ;do arquivo-1
          ldir                             ;preenche o nome com
                                           ;espaços
          exx                              ;recupera os registros
          ex      af,af'                   ;originais
          ret                              ;e volta
```

```
;a rotina copyfcb copia a definição do FCB original para
;o FCB de cópia (fcb2)


copyfcb
                exx                             ;salva os registros
                ld      hl,fcb1                 ;hl aponta para fcb1
                ld      de,fcb2                 ;de aponta para fcb2
                ld      bc,#0024                ;bc=comprimento do FCB
                ldir                            ;copia
                exx                             ;recupera os registros
                ret                             ;retorna


mensagem_1
                defm    "Especificação inválida"
                defb    #0d,#0a,#0d,#0a
                defm    "Sintaxe: prog31 [Drive:] Nome.Ext"
                defb    #0d,#0a
                defm    "onde:"
                defb    #0d,#0a,#09
                defm    "[Drive:] é um parâmetro opcional"
                defb    #0d,#0a,#09
                defm    "Nome é o nome do arquivo"
                defb    #0d,#0a,#09
                defm    "Ext é a extensão do arquivo$"


mensagem_2
                defm    "Erro de leitura!$"


mensagem_3
                defm    "Erro de gravação!$"

mensagem_4
                defm    "Diretório sem espaço!$"

mensagem_5
                defm    "Disco sem espaço!$"

fcb1            defs    40
fcb2            defs    40
```

Para poder executar o programa acima, você deve primeiramente compilá-lo usando o GEN80. Após a etapa de compilação, o GEN80 criará o arquivo PROG31.COM, que você poderá executar. Para saber a sintaxe da linha de comando, entre somente com

PROG31

a partir do **A>**. Neste caso, o programa tentará abrir um arquivo que não foi especificado, o que por sua vez resultará num erro, levando o programa a exibir a sintaxe da linha de comando. Como todas as listagens apresentadas neste livro, esta apresenta um número suficiente de comentários para permitir a fácil compreensão de cada uma das rotinas utilizadas.

Com este programa, encerramos a nossa viagem pelo sistema de disco do MSX. No próximo capítulo, vamos aprender como acessar os drives diretamente através das portas do FDC (sigla, em inglês, do controlador de discos flexíveis). Interessante observar que a esmagadora maioria das cartas que recebi me pedia justamente as informações contidas no próximo capítulo.

BIBLIOGRAFIA RECOMENDADA

SISTEMAS OPERACIONAIS DO MSX E SUAS FERRAMENTAS - Sérgio Guy Pinheiro Elias e Paulo Roberto Pinheiro Elias - Rio de Janeiro - EDITORA CIÊNCIA MODERNA LTDA.

SISTEMA DE DISCO PARA MSX - EDITORA ALEPH

ROTINAS PRONTAS MSX - Richard Spiegel - Rio de Janeiro - LTC - Livros Técnicos e Científicos Editora Ltda.

DRIVE POR ENDEREÇOS

SHARP
MSX INTERNACIONAL

ENDEREÇO	PORTA
7FF8	= D0H
7FF9	= D1H
7FFA	= D2H
7FFB	= D3H
7FFD	= D4H (LUZES = { 0 = LIGA / 1 = DESLIGA)
7FFF	= ?

FDC REGISTER

WRITE	READ
COMMAND	STATUS
TRACK	TRACK
SECTOR W	SECTOR R
DATA	DATA
DRIVE	DRIVE

Capítulo 7

PROGRAMANDO O FDC

Este capítulo é especialmente dedicado a todos aqueles que possuem razoáveis conhecimentos da linguagem assembly no MSX e desejam aprender a operar o FDC diretamente, ou seja, sem a ajuda do BDOS ou BIOS. Antes de mais nada, o que é o FDC? O FDC (da terminologia inglesa **F**loppy **D**isk **C**ontroller - Controlador de Disco Flexível) é um processador dedicado às operações de I/O (Entrada/Saída) com os disk drives, ou seja, é o cérebro da sua interface de drive, da mesma forma que o Z 80 é o cérebro do seu MSX. Você já sabe que, para acessar um processador paralelo como o processador de vídeo, tem de fazer uso das instruções que lêem e escrevem em portas do Z 80 (em alguns computadores, principalmente nos mais antigos, este acesso pode ser feito por intermédio de endereços), nomeadamente as instruções **IN** e **OUT**. Para acessar o FDC, não será diferente.

A ESTRUTURA DO FDC

O FDC faz uso de cinco registros para as mais variadas funções. Para nós e para o Z 80, esses cinco registros nada mais são do que portas (como as portas do VDP) com funções específicas. O projeto MSX prevê a utilização de até 8 portas (#D0 a #D7) pelo FDC. As interfaces controladoras atualmente comercializadas no BRASIL fazem uso somente das cinco primeiras (#D0 a #D5) portas. Vejamos a função de cada uma delas.

PORTA	FUNÇÃO
#D0	Comando/Status
#D1	Trilha atual
#D2	Informar o setor
#D3	Enviar/receber dados
#D4	Seleção do drive

Tabela 7.1: Os registros do FDC

A porta #D0 é utilizada para informar ao FDC que comando desejamos que ele execute. Como exemplos, temos comandos para leitura de um setor, gravação de um setor, leitura de uma trilha, gravação de uma trilha, posicionamento do cabeçote, etc. Após a execução do re-

ferido comando, a porta #D0 passa a indicar o status da operação, ou seja, o modo como a execução procedeu (com erro, sem erro, etc.). Como você pode perceber, esta é a principal porta/registro do FDC.

A porta #D1 é utilizada para indicar a trilha atual. Este é, talvez, o registro mais discutido do FDC, já que não preserva o valor ao trocarmos de drive. Isto quer dizer que, se estivermos acessando o drive A e o registro de trilha (porta #D1) indicar que o cabeçote está na trilha 20, por exemplo, ao passarmos para o drive B este registro continuará indicando que o cabeçote está posicionado na trilha 20, embora o cabeçote do drive B possa estar em qualquer outra trilha. Vale lembrar que a numeração de trilhas se inicia em 0 e termina em 39.

A porta #D2 é utilizada para enviarmos o número do setor a ler/escrever dentro da trilha já selecionada. Aqui, cabe uma observação. A numeração dos setores numa trilha se inicia em 1 e termina em 9. Desta forma, todas as trilhas possuem a mesma numeração interna (1 a 9) para os setores. O FDC não reconhece um comando do tipo "leia o setor 719". O que ele reconhece é a seqüência de comandos equivalente, ou seja:

1.Posicione o cabeçote na trilha 39.
2.Selecione o segundo lado do disquete
3.Leia o setor 9 desse lado

Dá para perceber que há o envolvimento de um pequeno algebrismo, não? Mas não se desanime.

A porta #D3 é o registro de dados do FDC. Em qualquer operação de leitura ou de gravação, os bytes a serem lidos/gravados são lidos/enviados seqüencialmente de ou para esta porta. "O que isso quer dizer?" Simplesmente, que não existe a leitura de um setor completo, mas sim a leitura de cada um dos 512 bytes que o compõem. Esta porta também serve para enviarmos a nova trilha onde o cabeçote deverá se posicionar.

A porta #D4 é o registro do FDC que controla a seleção das unidades de disco. É interessante observar que este registro não possui um controle individual para os motores dos drives, isto é, quando damos ordem de ligar o motor do drive A, o motor do drive B também é acionado. Por outro lado, este registro faz a seleção do drive e da face do disquete dentro do referido drive.

Vamos passar agora ao estudo detalhado de cada registro do FDC.

O REGISTRO #D0

O registro #D0 possui dupla função:

1.Informar ao FDC que comando desejamos que ele execute, e

2.Retornar com informações sobre o andamento do último comando.

Devido às funções que desempenha, este é, sem dúvida, o principal registro do FDC. Vamos observar primeiro os comandos que ele oferece.

COMANDO	TIPO	FUNÇÃO
Restore	1	Posiciona o cabeçote na trilha 0
Seek	1	Posiciona o cabeçote na trilha especificada
Step	1	Move o cabeçote uma trilha na direção do último movimento
Step in	1	Move o cabeçote uma trilha para a frente
Step out	1	Move o cabeçote uma trilha para trás
Read sector	2	Lê um setor
Write sector	2	Grava um setor
Read address	3	Lê o byte de ID da trilha
Read track	3	Lê uma trilha
Write track	3	Grava uma trilha
Force interrupt	4	Força a parada da execução do último comando

Tabela 7.2: Os comandos do FDC

Cada comando acima é o resultado de um mapeamento por bits do valor enviado para a porta #D0. Vejamos o mapeamento por bits de cada um dos comandos acima.

COMANDO	BITS 7 6 5 4 3 2 1 0	VALOR TPICO (em hexa)
Restore	0 0 0 0 0 V r1 r0	#01
Seek	0 0 0 1 0 V r1 r0	#11
Step	0 0 1 u 0 V r1 r0	#31
Step in	0 1 0 u 0 V r1 r0	#51
Step out	0 1 1 u 0 V r1 r0	#71
Read sector	1 0 0 0 F2 F1 0	#80
Write sector	1 0 1 0 F2 F1 0	#A0
Read address	1 1 0 0 0 E 0 0	#C4
Read track	1 1 1 0 0 E 0 0	#E4
Write track	1 1 1 1 0 E 0 0	#F4
Force interrupt	1 1 0 1 0 0 0 0	#D0

Tabela 7.3: Mapeamento por bits dos comandos do FDC

A codificação usada nos comandos acima é a seguinte:

V - Flag de verificação (bit 2)

V = 1 - Verifica a trilha-destino
V = 0 - Não verifica

r1 e r0 - Retardo do motor de passo (bits 0 e 1)

r1	r0	
0	0	6 milissegundos
0	1	12 milissegundos (valor recomendado)
1	0	20 milissegundos
1	1	40 milissegundos

u - Flag de atualização do registro da trilha (bit 4)

u = 1 - Atualiza o registro da trilha
u = 0 - Não atualiza

E - Espera de 40 milissegundos (bit 2)

E = 1 - Espera 40 milissegundos
E = 0 - Não espera

F2 - Flag de seleção de lado (bit 3)

F2 = 0 - Compara pelo lado 0
F2 = 1 - Compara pelo lado 1

F1 - Flag de comparação (bit 1)

F1 = 1 - Habilita a comparação de lado
F1 = 0 - Desabilita a comparação de lado

Tabela 7.4: Codificação usada nos comandos do FDC

O comando STEP depende da utilização prévia do comando STEP IN ou STEP OUT para que o FDC saiba em que sentido deslocar o cabeçote. Tome cuidado ao usar estes comandos dentro dos limites de trilha (0 a 39 para drives de 5 1/4" e 0 a 79 para drives de 3 1/2"). Embora a maioria dos drives de 5 1/4" permita o acesso de 42 (0 a 41) trilhas, recomendo que você se limite ao padrão estabelecido de 40 trilhas.

O termo *motor de passo* se aplica a um pequeno motor de alta precisão que tem como função deslocar o cabeçote de leitura/gravação para frente e para trás, em incrementos precisos correspondentes ao salto de uma trilha para outra. Quando este motor se desregula, os saltos de uma trilha para outra não são mais respeitados, ocasionando erros de leitura/gravação. Assim sendo, podemos concluir que existem dois motores num drive: o motor de passo e o motor que rota o disquete à velocidade de 300 rpm (rotações por minuto). Não vamos nos deter em exemplos agora, já que o final deste capítulo apresenta vários deles sobre os comandos acima.

Vamos passar agora à codificação usada pelo FDC no registro #D0 para relatar o andamento de uma operação. Para verificação do tipo de comando, consulte a Tabela 7.2.

BIT	NOME	FUNÇÃO (quando em 1 - setado)
0	Ocupado	Comando em progresso
1	Índice	Encontrada a marca de índice
2	Trilha 0	Cabeçote posicionado sobre a trilha 0
3	Erro de CRC	Erro de soma em relação ao ID
4	Erro de busca	Erro de posicionamento do cabeçote
5	Cabeça	Não é usado nos drives do MSX. Só tem uso nos drives de 8".
6	Proteção	Indica que o disquete está protegido contra gravação
7	Não pronto	Indica que o drive ainda não atingiu a velocidade correta

Tabela 7.5: Status para os comandos dos tipos 1 e 4

BIT	NOME	FUNÇÃO (quando em 1 - setado)
0	Ocupado	Comando em progresso
1	DRQ	Data Request (requisição de dados) ativado na transferência de dados
2	Perda de dados	O computador não responde ao DRQ em tempo
3	Erro de soma	Se o bit 4 estiver setado, ocorreu erro no campo ID; se não, erro no campo de dados
4	Registro	Trilha, setor ou lado não encontrado
5	Erro	Erro de gravação
6	Proteção	Proteção contra escrita
7	Não pronto	O drive não atingiu a rotação correta

Tabela 7.6: Status para os comandos dos tipos 2 e 3

O termo *marca de índice* está relacionado ao pequeno furo, ao lado do furo central, existente nos disquetes de 5 1/4". Este pequeno furo tem como função detectar a posição do setor 0 numa trilha. Exatamente abaixo desse furo existe no drive uma pequena luz. Ao coincidir o furo da mídia (o disquete propriamente dito) com o furo da capa do disquete, a luz passa pelos dois e atinge um mecanismo sensível à ela, que por sua vez informa ao FDC a detecção da marca de índice. Desta forma, o FDC saberá que, a partir desse ponto, poderá acessar o setor 0 da trilha sobre a qual o cabeçote está posicionado. Vale lembrar que a proteção contra escrita utiliza um mecanismo semelhante, de modo que **não adianta proteger o seu disquete com fita adesiva transparente**. É interessante observar que os drives para disquetes de 3 1/2" só fazem uso do sistema ótico para detectar a proteção contra gravação. A detecção da marca de índice é feita por um pequeno ímã grudado no prato do motor de rotação, na parte inferior do drive. De um lado do drive existe uma pequena bobina sobre a qual a passagem do ímã induz uma pequena corrente que, então, ativa o circuito que detecta a marca de índice. Tudo isto se deve à filosofia do disquete de 3 1/2": um disquete lacrado que não apresenta nenhum orifício externo que permita a exposição da mídia.

Após o envio de cada comando, torna-se necessário verificar se ele já foi executado ou não. Isto é necessário porque o FDC depende de movimentos mecânicos, que possuem uma velocidade incomparavelmente menor do que a velocidade de execução de uma instrução em ASSEMBLY. Por outro lado, o próprio FDC não é tão rápido quanto o Z-80, o que nos obriga a provocar um retardo antes de começar a examinar o andamento da operação. Embora pareça que não, a sincronização desses tempos relativos é uma tarefa extremamente simples, como veremos nos exemplos do final deste capítulo. Não obstante, cabe lembrar que justamente nesta temporização residem os problemas de determinadas rotinas que deixam de funcionar.

O REGISTRO #D1

Este registro armazena o número da trilha atual.Como já vimos, o FDC não possui um registro deste para cada drive, o que nos obriga a manter uma tabela com o posicionamento atual dos respectivos cabeçotes. "Mas, por que tanto trabalho?" Rapidez! Como sabemos, uma trilha possui vários setores, logo, se o próximo setor a ser acessado estiver na mesma trilha que o anterior, para que perder tempo reposicionando o cabeçote? Não se esqueça que para ler um disco de 3 1/2" de face dupla existem 1440 setores e, se formos reposicionar o cabeçote para a leitura de cada setor, gastaremos tempo em 1440 reposicionamentos; já se aproveitarmos o esquema de acompanhamento por tabela, gastaremos o tempo necessário para fazer apenas 80 (número de trilhas) reposicionamentos. É um método muito mais inteligente, não?

O registro #D1 serve tanto para leitura (IN) como para escrita (OUT). Pelos motivos expostos anteriormente, não nos interessa muito a leitura de tal registro, porém, para reposicinar o cabeçote numa nova trilha precisamos antes informar ao FDC o número dela, enviando-o por esta porta/registro esse número.

O REGISTRO #D2

Embora possa ser usado para leitura do setor atual, este registro só tem sentido para o envio do número do setor (1 a 9) que queremos acessar dentro da trilha já especificada, ou a ser especificada. A única observação cabe ao número do setor, que se inicia em 1 e termina em 9, ao contrário de comandos em BASIC que acessam o setor 0.

O REGISTRO #D3

Este registro é a porta utilizada para a transferência de dados entre o FDC e o Z-80. Além de ser utilizado na leitura/gravação de setores e trilhas, este registro também serve para enviarmos o número da trilha sobre a qual desejamos posicionar o cabeçote, usando o comando SEEK. Durante as leituras ou gravações de trilhas ou de setores, os bytes são transferidos um a um por intermédio desta porta. Isto quer dizer que, se estivermos lendo um setor, teremos de fazer 512 leituras na porta.

O REGISTRO #D4

Este último registro é o responsável pelas seleções do drive, do lado do disquete e pelo acionamento dos motores dos drives. Vejamos a codificação empregada neste comando.

BIT	FUNÇÃO
0	Drive A (0)
1	Drive B (1)
2	Drive C (2)
3	Drive D (3)
4	Seleção da face do disquete (0 ou 1)
5	Acionamento dos motores
6	Habilita espera de dados
7	Densidade:0=simples, 1=dupla

Tabela 7.7: Codificação do registro #D4

O bit 7 não tem função definida no MSX, de modo que podemos deixá-lo sempre com o valor 0. O bit 6 deve estar sempre setado (em 1) para podermos acessar os drives de 3 1/2", que são, em geral, mais lentos que os de 5 1/4". O bit 5, quando setado, aciona os motores de todos os drives conectados a uma mesma interface. Da mesma forma, quando o bit 5 é colocado em 0 (resetado), provoca a parada desses motores. O bit 4 é o que apresenta uma lógica de acesso mais complexa. Para entender quando colocá-lo em 1 e quando colocá-lo em 0, precisamos seguir algumas etapas prévias, começando pelo cálculo da trilha e do setor den-

tro da trilha, baseado no número do setor passado na forma tradicional (de 0 a 719, no caso de disquete de 5 1/4" de face dupla, e de 0 a 1439, no caso de disquete de 3 1/2" de face simples), conforme veremos abaixo. Os bits 0 a 3 selecionam (operação denotada pela pequena luz na parte frontal do drive) o drive a ser acessado. Se você quiser fazer experiências, entre com o seguinte programa em BASIC:

```
10 PRINT "ACIONANDO O DRIVE A ..."
20 OUT &HD4,&B00100001
30 FOR A%=1 TO 5000:NEXT A%
40 PRINT "PARANDO O DRIVE A E MANTENDO A SELEÇÃO..."
50 OUT &HD4,&B00000001
60 FOR A%=1 TO 5000:NEXT A%
70 PRINT "TERMINANDO A SELEÇÃO DO DRIVE A..."
80 OUT &HD4,&B00000000
```

CÁLCULO DA TRILHA E DO SETOR

O cálculo da trilha e do setor dentro dessa trilha não é tão complicado quanto possa parecer. Como sabemos, um disquete possui 9 setores por trilha em cada lado (não esquecer que nos disquetes de face dupla existe uma trilha de cada lado). O que temos de fazer, então, é uma divisão do número do setor passado pelo número de setores por trilha. Vamos exemplificar para tornar mais claro. Suponha que você deseje acessar o setor 182 de um disquete de 5 1/4" de face simples (180 Kb). As contas seriam as seguintes:

Número do setor:	182
Número de setores por trilha:: :	9
Trilha a acessar :	INT (182/9)=20
Setor a acessar :	182-(INT(182/9))*9+1=3

Logo, teríamos de acessar o setor 3 da trilha 20 no lado 0, já que o disquete de face simples não apresenta lado 1. Note que a expressão

182-(INT(182/9))*9

é o resto da divisão **182/9**. Como esse resto estará sempre na faixa de 0 a 8, temos de somar 1 para colocá-lo na faixa de 1 a 9, que vem a ser a faixa usada pelo FDC na numeração dos setores (na verdade, o FDC poderia numerar de 0 a 9 ou de 10 a 19, ou ainda em qualquer outra faixa que você bem entenda, mas o padrão MS-DOS estabelece que a numeração deve estar entre 1 e 9). Foi difícil? Claro que não! "Mas, e se o disquete fosse de face dupla?" Nesse caso, teríamos de apelar para um macete. Observe que o maior setor possível num disquete de face simples é o 359 (360 setores numerados de 0 a 359), o que implica afirmar que a divisão acima resultará em valores para trilha entre 0 e 39, que vem a ser exatamente o número de trilhas no disquete (40 trilhas numeradas de 0 a 39). No disquete de face dupla (ainda

de 5 1/4"), o maior setor é o 719 (720 setores numerados de 0 a 719), logo, o resultado da divisão acima resultará em valores para trilha entre 0 e 79. Como sabemos, o disquete de face dupla só possui 40 trilhas de cada lado. Então, o que fazer? Vamos tomar dois exemplos: achar a trilha e o setor nas trilhas do setor 7 e do setor 15.

Setor 7:

Número do setor:	7
Número de setores por trilha :	9
Trilha a acessar :	INT (7/9)=0
Setor a acessar :	7-(INT(7/9))*9+1=8

Setor 15:

Número do setor:	15
Número de setores por trilha :	9
Trilha a acessar :	INT (15/9)=1
Setor a acessar :	15-(INT(15/9))*9+1=7

Como sabemos,o setor 7 está no lado 0 da trilha 0, e o setor 15 no lado 1 da trilha 0. O que o resultado nos informou é que o setor 7 está na trilha 0 e o setor 15 na trilha 1. Parece estranho, não? Vamos ver mais um exemplo: do setor 25 e do setor 33.

Setor 25:

Número do setor:	25
Número de setores por trilha :	9
Trilha a acessar :	INT (25/9)=2
Setor a acessar :	25-(INT(25/9))*9+1=8

Setor 33:

Número do setor:	33
Número de setores por trilha	:9
Trilha a acessar :	INT (33/9)=3
Setor a acessar :	33-(INT(33/9))*9+1=7

Os resultados acima já revelaram mais algumas coisas:

1.O setor 15 fica no lado 1 da trilha 0, e não na trilha 1.

2.O setor 25 fica no lado 0 da trilha 1, e não na trilha 2.

3.O setor 33 fica no lado 1 da trilha 1, e não na trilha 3.

Você reparou que os setores que ficam no lado 1 de uma trilha dão como resultado trilhas ímpares (1 e 3 para os setores 15 e 33, respectivamente)? Você também reparou que os valores atribuídos às trilhas correspondem ao dobro dos valores reais (1 para o setor 15, 2 para o setor 25 e 3 para o setor 33)? Note que INT (1/2) = 0, INT (2/2) = 1 e INT (3/2) = 1. O que podemos concluir, então? Simplesmente que sempre que o resultado para a trilha for ímpar, devemos selecionar o lado 1 do disquete. Por outro lado, devemos sempre dividir por dois o valor achado para a trilha. "Mas, como saber, em assembly, se um determinado valor é ímpar ou par?" Basta examinar o bit 0! Este é o único bit que fornece a unidade: $2^0=1$, $2^1=2$, $2^2=4$, $2^3=8$, $2^4=16$, $2^5=32$, $2^6=64$ e $2^7=128$. Para o caso dos disquetes de 3 1/2", repetem-se exatamente os mesmos raciocínios, o que nos permite desenvolver um algoritmo geral para todos os tipos de disquete do MSX-DOS.

Para ilustrar as informações acima, vamos examinar um programa que faz a chamada formatação lógica de setores.

O PROGRAMA PARA A FORMATAÇÃO LÓGICA DE SETORES

Este programa tem como objetivo formatar logicamente, ou seja, sem perder as informações, um determinado setor que esteja apresentando erro de leitura (na verdade, erro de CRC), que no BASIC recebe o nome de erro de E/S. Por se tratar de um programa de demonstração, não leva em conta certas ocorrências, como a possibilidade do disquete estar protegido contra gravação, etc., embora faça o teste para cada uma das hipóteses possíveis. Fica a seu cargo implementá-lo. Uma boa parte das rotinas já foi utilizada nos capítulos anteriores, ficando a novidade com as rotinas de acesso direto ao FDC para leitura e gravação de setores. Vale lembrar que **as rotinas aqui apresentadas funcionam tanto em drives de 5 1/4" como de 3 1/2" que utilizem a interface padrão MICROSOL (todas, com a exceção da interface da SHARP)**.

Listagem em assembly Z-80 do código-fonte do programa para formatação lógica:

```
;programa para formatação lógica
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG32.BIN=PROG32.GEN
;
;onde PROG32.GEN é o nome do arquivo-texto com esta listagem
```

```
versao          equ     #002d
bufset          equ     #f351
txtnam          equ     #f3b3
valtyp          equ     #f663
argusr          equ     #f7f8
scrmod          equ     #fcaf

        defb    #fe             ;simula em CP/M
        defw    inicio          ;o cabecalho de
        defw    fim             ;um arquivo
        defw    inicio          ;.BIN

                org     #d000

inicio
                ld      a,(scrmod)      ;obtém o modo da tela
                or      a               ;modo texto?
                ret     nz              ;Não, volta
                ld      a,(valtyp)      ;parâmetro inteiro?
                cp      #02
                ret     nz              ;Não, volta
                ld      a,(argusr)      ;Drive > B
                cp      #03             ;
                ret     nc              ;Sim, volta
                or      a               ;é zero?
                ret     z               ;Sim, volta
                di                      ;desabilita as interrupções
                ld      ix,pardrive     ;ix aponta para
                                        ;a tabela com os
                                        ;parâmetros do
                                        ;drive
                call    leboot          ;Lê o boot para
                                        ;saber o tipo de
                                        ;disquete
                ld      a,"0"           ;
                call    tela            ;constrói a primeira
                                        ;tela
                call    exame_1         ;examina os setores
                ld      a,(ix+#03)      ;obtém o tipo de
                                        ;formatação
                cp      #fa             ;é de 80 trilhas
                jr      nc,fimrot       ;Não, termina
                ld      a,"4"           ;Sim, constrói
                call    tela            ;a segunda
                                        ;tela
```

```
                call    exame_2         ;examina os setores

fimrot
                call    rodadrive       ;para o drive na trilha 0
                ei                      ;habilita as interrupções
                ret                     ;e retorna



;a rotina exame_1 faz o exame das primeiras 40 trilhas do
;disquete

exame_1
                ld      hl,#0000        ;hl=setor inicial
                ld      (setatual),hl   ;salva
                jr      exame           ;vai para exame


;a rotina exame_2 faz o exame das ultimas 40 trilhas num
;disquete de 80 trilhas

exame_2
                ld      hl,720          ;hl=setor inicial
                ld      (setatual),hl   ;salva


;a rotina exame faz o exame do disquete propriamente dito

exame
                ld      hl,#0005        ;X=0 Y=5
                ld      b,40            ;b=número de trilhas
exame_3                 push    bc      ;salva b
                push    hl              ;salva as coordenadas
                ld      a,(ix+#02)      ;a=número de lados
                ld      b,a             ;b=número de lados
exame_4         push    bc              ;salva b
                ld      b,9             ;b=9 setores por trilha
exame_5         push    bc              ;salva b
                push    hl              ;salva as coordenadas
                ld      de,(bufset)     ;faz a leitura
                ld      hl,(setatual)   ;
                ld      b,#01           ;
                call    lesetor         ;lê o setor
                jr      nc,exame_6      ;Se não houve erro,
                                        ;pula para exame_6
                ld      de,(bufset)     ;Se houve, tenta
                ld      hl,(setatual)   ;regravar o setor
```

```
            ld      b,#01           ;
            call    gravsetor       ;grava o setor
            jr      nc,exame_6      ;Se conseguiu, pula para
                                    ;exame_6. Se não,
            ld      (setatual),hl   ;salva o próximo setor
            pop     hl              ;informa que o
            push    hl              ;setor não pode mais
            call    posit           ;ser recuperado
            ld      a,"I"           ;(I)rrecuperável
            out     (#98),a         ;
            jr      exame_7         ;pula para exame_7
exame_6
            ld      (setatual),hl   ;salva o novo setor
            pop     hl              ;coloca um quadrado
            push    hl              ;indicando tudo ok.
            call    posit           ;
            ld      a,#db           ;
            out     (#98),a         ;
exame_7
            pop     hl              ;recupera as coordenadas
            inc     l               ;incrementa a linha
            pop     bc              ;repete para o próximo setor
            djnz    exame_5         ;
            pop     bc              ;repete para o próximo lado
            djnz    exame_4         ;se existir,
            pop     hl              ;recupera as coordenadas
            inc     h               ;aponta para a próxima trilha
            pop     bc              ;repete para as demais
                                    ;trilhas
            djnz    exame_3         ;
            ret                     ;sai do exame


;a rotina leboot lê os setores 0 do disquete-fonte e de
;destino para comparar os tipos e obter o número de
;setores a copiar. O parâmetro da função USR passa
;o número do drive a analisar

leboot
            ld      hl,512          ;inicializa o número de
            ld      (numbytes),hl   ;bytes por setor
            ld      a,#09           ;envia o número de setores
            ld      (numsetor),a    ;por trilha
            ld      a,(argusr)      ;obtém o drive
            ld      (drive),a       ;
            or      #20             ;soma com o parâmetro
```

```
                                        ;para acioná-lo
               ld      (ix+#00),a       ;
               call    rodadrive        ;coloca na trilha 0
               ld      de,(bufset)      ;obtém o endereço
                                        ;do buffer
               ld      hl,#0000         ;hl=núm. do setor
                                        ;a ler
               ld      b,#01            ;b=número de setores
                                        ;a ler
               call    lesetor          ;lê o setor
               ld      iy,(bufset)      ;obtém o número de bytes
               ld      l,(iy+#0b)       ;por setor
               ld      h,(iy+#0c)       ;
               ld      (numbytes),hl    ;
               ld      a,(iy+#18)       ;obtém o número de setores
               ld      (numsetor),a     ;por trilha
               ld      a,(iy+#15)       ;obtém o tipo de formatação
               push    af               ;salva na pilha
               ld      a,(iy+#1a)       ;obtém o número de lados
               ld      (ix+#02),a       ;ajusta a tabela
               pop     af               ;recupera o tipo
               ld      (ix+#03),a       ;ajusta a tabela
               ld      (ix+#01),#00     ;ajusta a trilha
               ret                      ;volta ao BASIC


;a rotina inidrive prepara o drive para leitura
;ou gravação de setores. Esta rotina calcula a
;trilha e setor onde posicionar o cabeçote

inidrive
               push    de               ;salva o end. do buffer
               call    testcond         ;espera liberação
               ld      a,(numsetor)     ;a=núm. de setores por trilha
               ld      c,a              ;c=núm. de setores por trilha
               ld      b,#08            ;prepara a divisão em 8 bits
divide_1       add     hl,hl            ;do número do setor desejado
               ld      a,h              ;pelo número de setores
               sub     c                ;por trilha
               jr      c,divide_2
               ld      h,a
               inc     l
divide_2       djnz    divide_1
               inc     h                ;h=núm. do setor (1 a 9)
               ld      a,(ix+#00)       ;parâmetro para ligar
               ld      e,a              ;salva em e
               ld      a,(ix+#02)       ;obtém o núm. de lados
```

```
                dec     a               ;face simples?
                ld      a,e             ;
                jr      z,inidrive_1    ;Sim, vai para inidrive_1
                srl     l               ;divide a trilha por 2
                                        ;o número da trilha
                                        ;é ímpar (lado 1)?
                jr      nc,inidrive_1   ;Não, vai para inidrive_1
                or      #10             ;Sim, muda o parâmetro para
                                        ;acessar o lado 1

inidrive_1
                ld      d,a             ;d=seleção
                ld      a,(ix+#03)      ;a=tipo de form.
                rrca                    ;
                rrca                    ;
                and     #c0             ;
                or      d               ;nova seleção
                ld      a,d             ;salva em d
                out     (#d4),a         ;liga o motor do drive
                call    testcond        ;espera liberação
                ld      a,h             ;envia o setor
                out     (#d2),a         ;
                call    testcond        ;espera liberação
                ld      a,(ix+#01)      ;a=trilha anterior
                out     (#d1),a         ;envia o número
                cp      l               ;já está na nova trilha?
                jr      z,acesetor      ;Sim, vai para inidrive_2
                ld      a,l             ;Não, posiciona o
                out     (#d3),a         ;cabeçote na trilha
                ld      (ix+#01),a      ;atualiza a variável
                ld      a,#11           ;
                out     (#d0),a         ;
                call    testcond        ;espera liberação
                ld      b,#10           ;
                call    espera          ;

;a rotina acesetor serve aos propósitos de leitura e/ou
;gravação de um setor. Na entrada, de=buffer em RAM e
;hl=número do setor. Os labels drive, numsetor e densid
;devem estar corretamente preenchidos

acesetor
                pop     hl              ;recupera o ptr. p/ buffer
                ld      e,5             ;e=núm. de tentativas
acesetor1       push    hl              ;salva o end. do buffer
```

```
               push    de              ;salva núm. de tentativas
               call    testcond        ;espera liberação
acesetor2      ld      a,#80           ;a=comando de leitura
               bit     6,d             ;espera ativada?
               jr      z,acesetor3     ;Não, vai p/ acesetor3
               or      #02             ;habilita comparação
               bit     4,d             ;é lado 1?
               jr      z,acesetor3     ;Não, vai para acesetor3
               or      #08             ;Sim, faz a comparação
                                       ;pelo lado 1
acesetor3      out     (#d0),a         ;envia o comando
               ld      b,#03           ;laço de espera para
               call    espera          ;sincronização
               ld      c,#d3           ;a=porta de dados do FDC
               ld      de,fimacesso    ;de=end. final
               push    de              ;salva end. final na pilha
acesetor4      in      a,(#d0)         ;lê status do FDC
               rrca                    ;acabou a leitura?
               ret     nc              ;Sim, vai para fimacesso
               rrca                    ;Não, espera chegar um byte
               jp      nc,acesetor4
acesetor5      ini                     ;lê/escreve (outi) um byte
               jp      acesetor4       ;prepara nova leitura/escrita

fimacesso
               pop     de              ;recupera núm. de tentativas
               pop     hl              ;recupera end. do buffer
               in      a,(#d0)         ;obtém o status do FDC
acesetor6      and     #9c             ;correu tudo bem?
               jr      z,saiacesso     ;Sim, volta
               bit     6,a             ;No caso de escrita, o disco
               jr      nz,comerro      ;estava protegido?
               dec     e               ;faz mais uma tentativa
               jr      nz,acesetor1
               bit     4,a             ;houve erro de busca?
               jr      nz,comerro      ;Sim, vai para comerro
               bit     3,a             ;houve erro de CRC?
               jr      nz,comerro      ;Sim, vai para comerro
                                       ;Não, então houve perda de
                                       ;dados
comerro
               ld      a,#01           ;indica erro
saiacesso
               call    prepfim         ;prepara a saída
               ret                     ;e volta

;a rotina prepfim prepara a saída da última operação
```

```
prepfim
                push    af                  ;salva af
                ld      a,#d0               ;força o término
                out     (#d0),a             ;
                call    testcond            ;espera liberar
                pop     af                  ;recupera af
                ret                         ;e volta


;rotina para leitura de um setor. hl=núm. do setor, de=end.
;em RAM do buffer para o setor

lesetor
                ld      iy,parleitura       ;iy aponta para os parâmetros
                                            ;de leitura
                ld      a,(iy+#00)          ;a=comando de leitura
                ld      (acesetor2+1),a     ;modifica a rotina acesetor
                ld      a,(iy+#01)          ;a=instrução ini (leitura)
                ld      (acesetor5+1),a     ;modifica a rotina acesetor
                ld      a,(iy+#02)          ;a=parâmetro de verificação
                                            ;da leitura
                ld      (acesetor6+1),a     ;modifica a rotina acesetor
lesetor_1
                push    hl                  ;salva hl
                push    bc                  ;salva bc
                push    de                  ;salva de
                call    inidrive            ;faz a leitura de um setor
                pop     de                  ;recupera o buffer
                ld      hl,(numbytes)       ;soma com o número de bytes
                add     hl,de               ;por setor
                ex      de,hl               ;
                pop     bc                  ;recupera os outros registros
                pop     hl                  ;
                inc     hl                  ;aponta para o próximo setor
                djnz    lesetor_1           ;lê os outros setores
                or      a                   ;houve erro?
                ret     z                   ;Não, volta.
                scf                         ;Sim, sinaliza
                ret                         ;e retorna

parleitura
                defb    #80,#a2,#9c


;rotina para a gravação de um setor. hl=núm. do setor,
;de=end. em RAM do buffer do setor
```

```
gravsetor
            ld      iy,parescrita       ;iy aponta para os parâmetros
                                        ;de escrita
            ld      a,(iy+#00)          ;a=comando de escrita
            ld      (acesetor2+1),a     ;modifica a rotina acesetor
            ld      a,(iy+#01)          ;a=instrução outi (escrita)
            ld      (acesetor5+1),a     ;modifica a rotina acesetor
            ld      a,(iy+#02)          ;a=parâmetro de verificação
            ld      (acesetor6+1)       ,a;da escrita
gravset_1
            push    hl                  ;salva os registros
            push    bc                  ;
            push    de                  ;
            call    inidrive            ;grava um setor
            pop     de                  ;recupera o buffer
            ld      hl,(numbytes)       ;soma com o número de bytes
            add     hl,de               ;por setor
            ex      de,hl               ;
            pop     bc                  ;recupera os outros registros
            pop     hl                  ;
            inc     hl                  ;aponta para o próximo setor
            djnz    gravset_1           ;grava os outros setores
            or      a                   ;houve erro?
            ret     z                   ;Não, volta
            scf                         ;Sim, sinaliza
            ret                         ;e retorna

parescrita
            defb    #a0,#a3,#fc


pardrive
            defb    #21                 ;parâmetro p/ ligar
            defb    #00                 ;trilha atual
            defb    #01                 ;densidade
            defb    #f9                 ;tipo de formatação


;rotina que pára o motor do drive sem mexer no cabeçote

paradrive
            push    af                  ;salva o registro afetado
            xor     a                   ;zera o registro a
            out     (#d4),a             ;envia o comando de parada
            pop     af                  ;recupera o registro
            ret                         ;retorna
```

```
;rotina para deslocar o cabeçote até a trilha zero e
;parar o drive. Esta rotina deve ser usada sempre que
;se desejar parar o drive.

rodadrive
                push    af              ;salva os registros afetados
                push    bc
                push    de
                push    hl
                ld      a,(drive)       ;obtém o drive
                ld      e,#20           ;comando para ligar o motor
                or      e               ;a=comando para acionar
                out     (#d4),a
                call    testcond        ;espera liberação
                ld      a,#02           ;a=comando de inicialização
                out     (#d0),a         ;envia o comando
                call    testcond        ;espera liberação
                ld      b,200           ;dá um tempo para o ajuste
                call    espera          ;das partes mecânicas
                call    paradrive       ;pára o motor
                pop     hl              ;recupera os registros
                pop     de
                pop     bc
                pop     af
                ret                     ;retorna


;rotina para estabilização das partes mecânicas do drive

espera
                ex      (sp),hl         ;retardo para sincronização
                ex      (sp),hl         ;
                djnz    espera          ;
                ret                     ;retorna


;rotina para testar a liberação do FDC

testcond
                ex      (sp),hl         ;retardo para sincronização
                ex      (sp),hl         ;
                ex      (sp),hl         ;
                ex      (sp),hl         ;
testcond_1
                in      a,(#d0)         ;
                rrca                    ;
                jr      c,testcond_1    ;
```

```
                ret


;a rotina tela constrói uma imagem gráfica
;das trilhas e setores do disquete em
;questão

tela
                exx                             ;salva os registros
                                                ;bc,de e hl
                ld      hl,#0003                ;X=0 Y=3
                call    posit                   ;posiciona em X,Y
                ld      b,#04                   ;constrói as linhas
tela_1          push    bc                      ;que indicam as
                out     (#98),a                 ;trilhas
                push    af                      ;
                ld      b,#09                   ;
                ld      a," "                   ;
tela_2          out     (#98),a                 ;
                ex      (sp),hl                 ;demora para
                ex      (sp),hl                 ;sincronização
                djnz    tela_2                  ;
                pop     af                      ;
                inc     a                       ;
                pop     bc                      ;
                djnz    tela_1                  ;
                ld      b,#04                   ;
tela_3          push    bc                      ;
                ld      a,"0"                   ;
                ld      b,10                    ;
tela_4          out     (#98),a                 ;
                inc     a                       ;
                ex      (sp),hl                 ;demora para
                ex      (sp),hl                 ;sincronização
                djnz    tela_4                  ;
                pop     bc                      ;
                djnz    tela_3                  ;
                ld      a,(ix+#02)              ;obtém o núm. de lados
                ld      b,a                     ;b=núm. de lados
tela_5          push    bc                      ;preenche a tela
                ld      bc,360                  ;com 360 pontos
tela_6          ld      a,"."                   ;um para cada setor
                out     (#98),a                 ;num lado do disquete
                dec     bc                      ;de 40 trilhas
                ld      a,b                     ;
                or      c                       ;terminou?
                jr      nz,tela_6               ;Não, repete.
```

```
                pop     bc              ;Sim. Se o disco for
                djnz    tela_5          ;face dupla, preenche
                                        ;mais 360 pontos
                exx                     ;recupera os registros
                ret                     ;e retorna


;a rotina posit posiciona o cursor nas coordenadas passadas
;por hl. h=x e l=y

posit
                push    af              ;salva todos os
                push    bc              ;registros afetados
                push    de              ;por esta rotina
                push    hl              ;
                ex      de,hl           ;troca o conteúdo de hl pelo
                                        ;de de
                ld      hl,#0000        ;zera hl
                ld      a,e             ;a=linha
                or      a               ;é igual a zero?
                jr      z,posit2        ;Sim, vai para posit2
                push    de              ;Não, salva as coordenadas
                ld      b,a             ;b=núm. da linha
                ld      a,(colunas)     ;a=núm. de colunas
                ld      e,a             ;e=núm. de colunas
                ld      d,#00           ;de=núm. de colunas
posit1          add     hl,de           ;hl=hl+núm. de colunas
                djnz    posit1
                pop     de              ;recupera as coordenadas
posit2          ld      a,d             ;a=coluna
                or      a               ;é igual a zero?
                jr      z,posit3        ;Sim, vai para posit3
                ld      e,a             ;Não, e=coluna
                ld      d,#00           ;de=coluna
                add     hl,de           ;hl aponta para o end. da
                                        ;VRAM
                                        ;correspondente a x1,y1
posit3          call    setvdpwt        ;prepara o VDP para escrita
                pop     hl              ;recupera os registros
                pop     de              ;
                pop     bc              ;
                pop     af              ;
                ret                     ;e retorna
;rotinas para o acesso direto à RAM de vídeo

rdvram
                call    setvdprd        ;ajusta o VDP para leitura
```

```
                in      a,(#98)         ;lê um byte
                ret                     ;retorna


wtvram
                push    af              ;salva o dado a enviar
                call    setvdpwt        ;ajusta o VDP para escrita
                pop     af              ;recupera o dado a enviar
                out     (#98),a         ;envia
                ret                     ;retorna

setvdprd
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,rdvram1       ;Sim, vai para rdvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a         ;
rdvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde será
                and     #3f             ;lido o dado
                out     (#99),a         ;
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                ret                     ;retorna


setvdpwt
                ld      a,(versao)      ;obtém a versão do MSX
                or      a               ;é MSX1?
                jr      z,wtvram1       ;Sim, vai para wtvram1
                xor     a               ;Não, inicializa o VDP
                out     (#99),a         ;do MSX2
                ld      a,#8e
                out     (#99),a
wtvram1         ld      a,l             ;informa ao
                out     (#99),a         ;VDP o endereço na
                ld      a,h             ;VRAM onde o
                and     #3f             ;dado será
                or      #40             ;gravado
                out     (#99),a         ;
                ex      (sp),hl         ;demora para
                ex      (sp),hl         ;sincronização
                ret                     ;retorna
```

```
;área das variáveis usadas no código-fonte

setatual        defw    #0000
numbytes        defw    #0000
numsetor        defb    #00
numset          defb    #00
drive           defb    #00
vez             defb    #00
colunas         defb    40      ;número de colunas na tela

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa para formatação lógica:

```
10 FOR A%=&HD000 TO &HD2DB
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG32.BIN",&HD000,&HD2DB
100 DATA 3A,AF,FC,B7,C0,3A,63,F6,FE,02,C0,3A,F8,F7,FE,03
110 DATA D0,B7,C8,F3,DD,21,EB,D1,CD,99,D0,3E,30,CD,27,D2
120 DATA CD,37,D0,DD,7E,03,FE,FA,30,08,3E,34,CD,27,D2,CD
130 DATA 3F,D0,CD,F5,D1,FB,C9,21,00,00,22,D2,D2,18,06,21
140 DATA D0,02,22,D2,D2,21,05,00,06,28,C5,E5,DD,7E,02,47
150 DATA C5,06,09,C5,E5,ED,5B,51,F3,2A,D2,D2,06,01,CD,8F
160 DATA D1,30,1C,ED,5B,51,F3,2A,D2,D2,06,01,CD,BD,D1,30
170 DATA 0E,22,D2,D2,E1,E5,CD,6A,D2,3E,49,D3,98,18,0C,22
180 DATA D2,D2,E1,E5,CD,6A,D2,3E,DB,D3,98,E1,2C,C1,10,C3
190 DATA C1,10,BD,E1,24,C1,10,B2,C9,21,00,02,22,D4,D2,3E
200 DATA 09,32,D6,D2,3A,F8,F7,32,D8,D2,F6,20,DD,77,00,CD
210 DATA F5,D1,ED,5B,51,F3,21,00,00,06,01,CD,8F,D1,FD,2A
220 DATA 51,F3,FD,6E,0B,FD,66,0C,22,D4,D2,FD,7E,18,32,D6
230 DATA D2,FD,7E,15,F5,FD,7E,1A,DD,77,02,F1,DD,77,03,DD
240 DATA 36,01,00,C9,D5,CD,1D,D2,3A,D6,D2,4F,06,08,29,7C
250 DATA 91,38,02,67,2C,10,F7,24,DD,7E,00,5F,DD,7E,02,3D
260 DATA 7B,28,06,CB,3D,30,02,F6,10,57,DD,7E,03,0F,0F,E6
270 DATA C0,B2,7A,D3,D4,CD,1D,D2,7C,D3,D2,CD,1D,D2,DD,7E
280 DATA 01,D3,D1,BD,28,12,7D,D3,D3,DD,77,01,3E,11,D3,D0
290 DATA CD,1D,D2,06,10,CD,18,D2,E1,1E,05,E5,D5,CD,1D,D2
300 DATA 3E,80,CB,72,28,08,F6,02,CB,62,28,02,F6,08,D3,D0
310 DATA 06,03,CD,18,D2,0E,D3,11,68,D1,D5,DB,D0,0F,D0,0F
320 DATA D2,5B,D1,ED,A2,C3,5B,D1,D1,E1,DB,D0,E6,9C,28,11
330 DATA CB,77,20,0B,1D,20,C4,CB,67,20,04,CB,5F,20,00,3E
340 DATA 01,CD,85,D1,C9,F5,3E,D0,D3,D0,CD,1D,D2,F1,C9,FD
350 DATA 21,BA,D1,FD,7E,00,32,41,D1,FD,7E,01,32,64,D1,FD
```

```
360 DATA 7E,02,32,6D,D1,E5,C5,D5,CD,E4,D0,D1,2A,D4,D2,19
370 DATA EB,C1,E1,23,10,EF,B7,C8,37,C9,80,A2,9C,FD,21,E8
380 DATA D1,FD,7E,00,32,41,D1,FD,7E,01,32,64,D1,FD,7E,02
390 DATA 32,6D,D1,E5,C5,D5,CD,E4,D0,D1,2A,D4,D2,19,EB,C1
400 DATA E1,23,10,EF,B7,C8,37,C9,A0,A3,FC,21,00,01,F9,F5
410 DATA AF,D3,D4,F1,C9,F5,C5,D5,E5,3A,D8,D2,1E,20,B3,D3
420 DATA D4,CD,1D,D2,3E,02,D3,D0,CD,1D,D2,06,C8,CD,18,D2
430 DATA CD,EF,D1,E1,D1,C1,F1,C9,E3,E3,10,FC,C9,E3,E3,E3
440 DATA E3,DB,D0,0F,38,FB,C9,D9,21,03,00,CD,6A,D2,06,04
450 DATA C5,D3,98,F5,06,09,3E,20,D3,98,E3,E3,10,FA,F1,3C
460 DATA C1,10,ED,06,04,C5,3E,30,06,0A,D3,98,3C,E3,E3,10
470 DATA F9,C1,10,F1,DD,7E,02,47,C5,01,68,01,3E,2E,D3,98
480 DATA 0B,78,B1,20,F7,C1,10,F0,D9,C9,F5,C5,D5,E5,EB,21
490 DATA 00,00,7B,B7,28,0C,D5,47,3A,DA,D2,5F,16,00,19,10
500 DATA FD,D1,7A,B7,28,04,5F,16,00,19,CD,B8,D2,E1,D1,C1
510 DATA F1,C9,CD,A0,D2,DB,98,C9,F5,CD,B8,D2,F1,D3,98,C9
520 DATA 3A,2D,00,B7,28,07,AF,D3,99,3E,8E,D3,99,7D,D3,99
530 DATA 7C,E6,3F,D3,99,E3,E3,C9,3A,2D,00,B7,28,07,AF,D3
540 DATA 99,3E,8E,D3,99,7D,D3,99,7C,E6,3F,F6,40,D3,99,E3
550 DATA E3,C9,00,00,00,00,00,00,00,00,28,00
```

Listagem do programa de teste:

```
1000 KEYOFF:WIDTH 40:DEFINT A-Z
1010 BLOAD "PROG25.BIN",R:BLOAD "PROG32.BIN"
1020 DEFUSR=&HD000
1030 GOSUB 2000:REM *** ESCOLHE O DRIVE ***
1040 CLS:LOCATE 0,1:PRINT "DRIVE : ";CHR$(&H41+A)
1050 A=USR(A+1)
1060 BEEP
1070 IF INKEY$="" THEN 1070
1080 CLS:END
2000 CLS:S1$="DRIVE A":S2$="DRIVE B":S3$="ESCOLHA O DRIVE"
2010 X1=(40-LEN(S3$))/2:X2=X1+LEN(S3$):Y1=10:Y2=15
2020 LOCATE X1,Y1-2:PRINT S3$
2030 WINDOW X1-1,Y1,X2,Y2,1
2040 X1=(40-LEN(S1$))/2:Y1=Y1+2:Y2=Y2-2
2050 LOCATE X1,Y1:PRINT S1$
2060 LOCATE X1,Y2:PRINT S2$
2070 MENU X1,Y1,X1,Y2,LEN(S1$),1,A
2080 RETURN
```

COMENTÁRIOS SOBRE O PROGRAMA PARA FORMATAÇÃO LÓGICA

O programa acima implementa as rotinas para leitura e gravação de setores. Observe que, após o envio de qualquer comando, existe uma chamada à rotina **testcond** que, após um pequeno retardo provocado por quatro instruções EX (SP),HL, testa o bit 0 do registro de status (#D0) para verificar a liberação do FDC para novas instruções. Como você já sabe, o bit 0 do registro de status indica se a última operação ainda está em andamento, logo, basta verificá-lo para saber se o FDC já está liberado (bit 0 resetado). A rotina **testcond** é aquela que permanece em execução se, durante uma operação de leitura ou de gravação, você abrir a porta do drive. Na eventual necessidade de deslocar o cabeçote para uma nova trilha, existe uma chamada à rotina **espera**, que tem como objetivo criar um pequeno retardo e, assim, fornecer um tempo extra para que o mecanismo do drive se estabilize. Observe que empregamos exatamente a mesma rotina para a leitura e gravação de setores, com o intuito de economizar alguns bytes. Para tanto, as rotinas **lesetor** e **gravsetor** modificam a rotina **acesetor** antes de chamarem a rotina **inidrive e acesetor**. A rotina **inidrive** tem como principal tarefa calcular a trilha e o setor dentro dessa trilha, baseada no número do setor passado pelo par HL. O setor passado pelo par HL deve estar no mesmo formato usado pelo BASIC: de 0 a 719 para disquetes de 5 1/4" de face dupla e de 0 a 1439 para disquetes de 3 1/2" de face dupla, por exemplo. É importantíssimo observar que o programa promove a desabilitação das interrupções logo no início. Isto é absolutamente necessário, tanto para o acesso ao vídeo como para o acesso aos drives. Imagine o que aconteceria se, no momento da leitura de um byte pelo registro #D3, o Z-80 fosse desviado por uma interrupção. Haveria uma perda de sincronismo, com uma conseqüente perda de dados. Terrível, não? Portanto, **desabilite (DI) as interrupções sempre que acessar o FDC**. Agora, um conselho: utilize as rotinas exatamente como apresentadas! Essas rotinas já estão sendo testadas há mais de dois anos (desde o lançamento do HELLO) sem qualquer problema, então, por que reinventar? Observe atentamente cada instrução das rotinas acima para tirar todas as dúvidas.

Ao executar o programa de teste, você verá que um disquete de 5 1/4" de dupla face é testado em 17 segundos, e um disquete de face dupla de 3 1/2" em 40 segundos (o drive de 3 1/2" é cerca de 12% mais lento). Tomando o pior caso, temos um gasto de 40 segundos no carregamento de 720 Kbytes, o que pode ser considerado um tempo excelente. A partir de agora, você pode contar com estas rotinas para o acesso acelerado ao disk drive. Mas, já que essas rotinas estão tão rápidas, que tal modificar o nosso copiador de setores que utiliza a MEGA-RAM?

O NOVO COPIADOR DE SETORES QUE USA A MEGARAM

Este programa é, basicamente, uma evolução do programa 28, apresentado no Capítulo 5. A única diferença reside no fato de que, agora, a leitura e a gravação de setores são realizadas por rotinas de acesso direto ao FDC, e não mais por intermédio da BIOS. Como você poderá comprovar, existe um ganho de quase 63% (1,5 minuto contra 4 minutos na cópia de disquetes de 5 1/4" de face dupla) no tempo de cópia de um disquete inteiro. Vamos então às listagens.

Listagem em assembly Z-80 do código-fonte do programa para cópia de setores:

```
;programa para cópia de setores usando a MEGARAM
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;GEN80 PROG33.BIN=PROG33.GEN
;
;onde PROG33.GEN é o nome do arquivo-texto com esta
;listagem

rdslt       equ     #000c
wrslt       equ     #0014
enaslt      equ     #0024
rslreg      equ     #0138
wslreg      equ     #013b
bufset      equ     #f351
valtyp      equ     #f663
argusr      equ     #f7f8
exptbl      equ     #fcc1

            defb    #fe             ;simula em CP/M
            defw    inicio          ;o cabecalho de
            defw    fim             ;um arquivo
            defw    inicio          ;.BIN

            org     #d500

inicio
            di                      ;desabilita as interrupções
            call    rslreg          ;lê a configuração atual
            ld      (confatual),a   ;salva em confatual
            xor     a               ;a=slot 0
            ld      hl,exptbl       ;hl aponta para a tabela
                                    ;de expansões de slots
            ld      b,#04           ;b=número de slots
                                    ;principais
loopbusca
            ld      (slotatual),a   ;Salva o slot atual
            ld      (subsatual),a   ;nestes endereços
            bit     7,(hl)          ;o slot atual está
                                    ;expandido?
            jr      nz,slotsecund   ;Sim, faz a procura
                                    ;nos slots secundários
```

```
                call    testamega       ;Não, testa a presença da
                                        ;MEGARAM no slot principal
                                        ;atual
                jr      c,fimbusca      ;Se achou, vai para fimbusca
loopbus_1
                inc     hl              ;Se não, repete a busca no
                ld      a,(slotatual)   ;slot seguinte
                inc     a               ;
                djnz    loopbusca       ;


naoachou
                ld      hl,#ffff        ;prepara a sinalização
                                        ;de que não achou a
                                        ;MEGARAM
                jr      fimbus_1        ;e volta ao BASIC


fimbusca
                ld      a,(subsatual)   ;obtém o slot/subslot
                                        ;onde encontrou a
                                        ;MEGARAM
                ld      l,a             ;prepara a volta ao
                ld      h,#00           ;BASIC

fimbus_1        ld      a,#02           ;indica parâmetro
                ld      (valtyp),a      ;inteiro
                ld      (argusr),hl     ;e passa a indicação
                                        ;para o BASIC
                ld      a,(confatual)   ;a=config. atual
                call    wslreg          ;restabelece a
                                        ;config. atual
                ei                      ;habilita as interrupções
                ret                     ;retorna ao BASIC


;a rotina slotsecun promove uma busca da MEGARAM nos
;quatro possíveis slots secundários de um slot
;principal

slotsecund
                push    bc              ;salva os registros
                push    hl              ;afetados
                ld      e,#00           ;e=1o. slot secundário
                ld      b,#04           ;b=4 slots secundários
slotsec_1
                ld      a,e             ;a=núm do slot secund.
```

```
            rla                           ;coloca o número nos
            rla                           ;bits 2 e 3
            and       %00001100           ;obtém somente os bits
                                          ;2 e 3
            ld        e,a                 ;salva o resultado em e
            ld        a,(slotatual)       ;obtém o slot principal
            and       %00000011           ;certifica-se de estar
                                          ;entre 0 e 3
            or        e                   ;a=ID do slot e subslot
            set       7,a                 ;indica que ID contém
                                          ;identificação de subslot
            ld        (subsatual),a       ;salva neste endereço
            call      testamega           ;testa a presença da
                                          ;MEGARAM
            jr        c,fimslotsec        ;Achou, vai para fimslotsec
            inc       e                   ;Se não, repete a busca
            djnz      slotsec_1           ;no subslot seguinte

fimslotsec
            pop       hl                  ;recupera os registros
            pop       bc                  ;salvos
            jp        nc,loopbus_1        ;Se não achou, continua
                                          ;a busca noutro slot
            jr        fimbusca            ;Se achou, volta para
                                          ;o BASIC


;a rotina testamega testa a presença da MEGARAM
;num determinado slot/subslot

testamega
            push      bc                  ;salva os registros
            push      de                  ;afetados
            push      hl                  ;
            call      leslot              ;lê o conteúdo da
                                          ;pos. #4000 do slot ou do
                                          ;subslot em questão
            push      af                  ;salva o valor lido
            out       (#8e),a             ;prepara a MEGARAM para
                                          ;chaveamento
            ld        e,#00               ;chaveia a MEGARAM para
            call      escslot             ;a página 0
            in        a,(#8e)             ;prepara a MEGARAM para
                                          ;escrita
            ld        e,#eb               ;e=valor a escrever
            call      escslot             ;escreve
            out       (#8e),a             ;prepara a MEGARAM para
```

```
                                          ;chaveamento
                ld      e,#00             ;chaveia a MEGARAM para
                call    escslot           ;a página 0
                call    leslot            ;lê o valor
                cp      #eb               ;compara com o valor
                                          ;escrito
                jr      z,fimteste        ;Se achou, vai para fimteste

                pop     af                ;Se não, recupera o valor
                ld      e,a               ;original e o repõe
                call    escslot           ;
                xor     a                 ;Zera a flag de carry para
                                          ;indicar que não encontrou
                                          ;a MEGARAM
                jr      fimteste_1        ;
fimteste
                pop     af                ;Repõe o valor
                ld      e,a               ;original
                call    escslot           ;
                scf                       ;indica que encontrou a
                                          ;MEGARAM
fimteste_1
                pop     hl                ;recupera os
                pop     de                ;registros salvos
                pop     bc                ;
                ret                       ;e retorna


;a rotina leslot lê o conteúdo do endereço
;#4000 de qualquer slot/subslot

leslot
                ld      hl,#4000          ;hl aponta para o
                                          ;endereço a ler
                ld      a,(subsatual)     ;obtém o ID do slot
                                          ;ou subslot
                call    rdslt             ;lê o valor
                ret                       ;retorna com o valor
                                          ;lido no registro a


;a rotina escslot escreve o conteúdo do registro
;e no endereço #4000 de qualquer slot/subslot

escslot
                ld      hl,#4000          ;hl aponta para o
```

```
                                     ;endereço a ser
                                     ;modificado
          ld      a,(subsatual)      ;obtém o ID do slot
                                     ;ou subslot
          call    wrslt              ;escreve o valor
                                     ;contido no registro
                                     ;e
          ret                        ;retorna


;transfere um bloco de 16 Kbytes da RAM para a MEGARAM

ram_mega
          di                         ;desabilita as interrupções
          call    inimega            ;inicializa a MEGARAM
          in      a,(#8e)            ;prepara a MEGARAM
                                     ;para escrita
          ld      hl,#9400           ;hl=end. inicial
          ld      de,#4000           ;de=end. final
          ld      bc,#4000           ;bc=16 Kbytes
          ldir                       ;transfere
          out     (#8e),a            ;
          ld      a,(confatual)      ;a=config. original
          call    wslreg             ;habilita a config.
                                     ;original dos slots
          ei                         ;habilita as interrupções
          ret                        ;retorna ao BASIC


;transfere um bloco de 16 Kbytes da MEGARAM para a RAM

mega_ram
          di                         ;desabilita as interrupções
          call    inimega            ;inicializa a MEGARAM
          ld      hl,#4000           ;hl=end. inicial
          ld      de,#9400           ;de=end. final
          ld      bc,#4000           ;bc=16 Kbytes
          ldir                       ;transfere
          out     (#8e),a            ;
          ld      a,(confatual)      ;a=config. original
          call    wslreg             ;habilita a config.
                                     ;original dos slots
          ei                         ;habilita as interrupções
          ret                        ;retorna ao BASIC
```

```
;a rotina inmega inicializa a MEGARAM.
;O parâmetro da função USR do BASIC
;deve indicar a página da MEGARAM a ser chaveada

inimega
                call    rslreg          ;lê a configuração
                                        ;atual dos slots
                ld      (confatual),a   ;salva em confatual
                ld      a,(subsatual)   ;obtém o ID da MEGARAM
                ld      hl,#4000        ;habilita a página
                call    enaslt          ;1 do slot da MEGARAM
                out     (#8e),a         ;prepara a MEGARAM
                                        ;para chaveamento
                ld      a,(argusr)      ;obtém a página da
                                        ;MEGARAM
                ld      (#4000),a       ;chaveia
                inc     a               ;chaveia a próxima
                                        ;página no endereço
                ld      (#6000),a       ;#6000
                ret                     ;retorna


;a rotina leboot lê os setores 0 do disquete-fonte e de
;destino para comparar os tipos e obter o número de
;setores a copiar. O parâmetro da função USR passa
;o número do drive a analisar

leboot
                ld      a,(vez)         ;é a primeira vez?
                or      a               ;
                jr      nz,leboot_1     ;Não, vai para leboot_1
                ld      hl,512          ;inicializa o número de
                ld      (numbytes),hl   ;bytes por setor
                ld      a,#09           ;envia o número de setores
                ld      (numsetor),a    ;por trilha
leboot_1
                ld      a,(argusr)      ;salva o número
                ld      (drive),a       ;do drive
                call    rodadrive       ;coloca na trilha 0
                ld      de,(bufset)     ;obtém o endereço
                                        ;do buffer
                ld      hl,#0000        ;hl=núm. do setor
                                        ;a ler
                ld      b,#01           ;b=número de setores
                                        ;a ler
                call    lesetor         ;lê o setor
                call    paradrive       ;para o drive
```

```
        ld      a,(vez)         ;é a primeira vez?
        or      a               ;
        ret     nz              ;Não, volta
                                ;Sim, inicializa os
                                ;parâmetros
        ld      ix,(bufset)     ;obtém o número de bytes
        ld      l,(ix+#0b)      ;por setor
        ld      h,(ix+#0c)      ;
        ld      (numbytes),hl   ;
        ld      a,(ix+#18)      ;obtém o número de setores
        ld      (numsetor),a    ;por trilha
        ld      a,(ix+#15)      ;obtém o tipo de formatação
        push    af              ;salva na pilha
        ld      a,(ix+#1a)      ;obtém o número de lados
        push    af              ;salva na pilha
        ld      ix,pardrivea    ;ix parâmetros drive a
        ld      iy,pardriveb    ;iy parâmetros drive b
        pop     af              ;recupera o núm. de lados
        ld      (ix+#02),a      ;ajusta as tabelas
        ld      (iy+#02),a      ;
        pop     af              ;recupera o tipo
        ld      (ix+#03),a      ;ajusta as tabelas
        ld      (iy+#03),a      ;
        ld      (ix+#01),#00    ;ajusta a trilha
        ld      (iy+#01),#00    ;inicial de ambos
                                ;os drives
        ld      a,#01           ;modifica a flag vez
        ld      (vez),a         ;
        ret                     ;volta ao BASIC


;a rotina leconj lê um conjunto de até 32 setores (16 Kb)
;do disquete-fonte

leconj
        ld      de,#9400        ;de=end. inicial
        ld      hl,(argusr)     ;obtém o número
                                ;do setor inicial
        ld      a,(numset)      ;obtém o número
                                ;de setores a ler
        ld      b,a             ;coloca em b
        call    lesetor         ;lê o(s) setor(es)
        ret                     ;e retorna
```

```
;a rotina grconj grava um conjunto de até 32 setores
;(16 Kb) no disquete de destino

grconj
            ld      de,#9400        ;hl=end. inicial
            ld      hl,(argusr)     ;obtém o número
                                    ;do setor inicial
            ld      a,(numset)      ;obtém o número
                                    ;de setores a ler
            ld      b,a             ;coloca em b
            call    gravsetor       ;grava o(s) setor(es)
            ret                     ;e retorna


;rotina para posicionar o cabeçote do drive na trilha
;correta, preparando-o para a leitura ou escrita do
;setor especificado

inidrive
            push    de              ;salva o ptr. p/ buffer
            call    testcond        ;espera liberação
            ld      a,(numsetor)    ;a=núm. de setores por trilha
            ld      c,a             ;c=núm. de setores por trilha
            ld      b,#08           ;prepara a divisão em 8 bits
divide_1    add     hl,hl           ;do número do setor desejado
            ld      a,h             ;pelo número de setores
            sub     c               ;por trilha
            jr      c,divide_2
            ld      h,a
            inc     l
divide_2    djnz    divide_1
            inc     h               ;h=núm. do setor (1 a 9)
            ld      a,(ix+#00)      ;parâmetro para ligar
            ld      e,a             ;salva em e
            ld      a,(ix+#02)      ;obtém o núm. de lados
            dec     a               ;face simples?
            ld      a,e             ;
            jr      z,inidrive_1    ;Sim, vai para inidrive_1
            srl     l               ;divide a trilha por 2
                                    ;O número da trilha
                                    ;é ímpar (lado 1)?
            jr      nc,inidrive_1   ;Não, vai para inidrive_1
            or      #10             ;Sim, informa.


inidrive_1
            ld      d,a             ;d=seleção
```

```
                ld      a,(ix+#03)      ;a=tipo de form.
                rrca                    ;
                rrca                    ;
                and     #c0             ;
                or      d               ;nova seleção
                ld      a,d             ;salva em d
                out     (#d4),a         ;liga o motor do drive
                call    testcond        ;espera liberação
                ld      a,h             ;envia o setor
                out     (#d2),a         ;
                call    testcond        ;espera liberação
                ld      a,(ix+#01)      ;a=trilha anterior
                out     (#d1),a         ;envia o número
                cp      l               ;já está na nova trilha?
                jr      z,acesetor      ;Sim, vai para inidrive_2
                ld      a,l             ;Não, posiciona o
                out     (#d3),a         ;cabeçote na trilha
                ld      (ix+#01),a      ;atualiza a variável
                ld      a,#11           ;
                out     (#d0),a         ;
                call    testcond        ;espera liberação
                ld      b,#10           ;
                call    espera          ;


;a rotina acesetor serve aos propósitos de leitura e/ou
;gravação de um setor. Na entrada, de=buffer em RAM e
;hl=número do setor. Os labels drive, numsetor e densid
;devem estar corretamente preenchidos

acesetor
                pop     hl              ;recupera o ptr. p/ buffer
                ld      e,5             ;e=núm. de tentativas
acesetor1       push    hl              ;salva o end. do buffer
                push    de              ;salva núm. de tentativas
                call    testcond        ;espera liberação
acesetor2       ld      a,#80           ;a=comando de leitura
                bit     6,d             ;espera ativada?
                jr      z,acesetor3     ;Não, vai p/ acesetor3
                or      #02             ;habilita espera
                bit     4,d             ;é lado 1?
                jr      z,acesetor3     ;Não, vai para acesetor3
                or      #08             ;Sim, faz a comparação
                                        ;pelo lado 1
acesetor3       out     (#d0),a         ;envia o comando
                ld      b,#03           ;laço de espera para
                call    espera          ;sincronização
```

```
                ld      c,#d3           ;a=porta de dados do FDC
                ld      de,fimacesso    ;de=end. final
                push    de              ;salva end. final na pilha
acesetor4       in      a,(#d0)         ;lê status do FDC
                rrca                    ;acabou a leitura?
                ret     nc              ;Sim, vai para fimacesso
                rrca                    ;Não, espera chegar um byte
                jp      nc,acesetor4
acesetor5       ini                     ;lê/escreve (outi) um byte
                jp      acesetor4       ;prepara nova leitura/escrita

fimacesso
                pop     de              ;recupera núm. de tentativas
                pop     hl              ;recupera end. do buffer
                in      a,(#d0)         ;obtém o status do FDC
acesetor6       and     #9c             ;correu tudo bem?
                jr      z,prepfim       ;Sim, volta
                bit     6,a             ;No caso de escrita, o disco
                jr      nz,comerro      ;estava protegido?
                dec     e               ;faz mais uma tentativa
                jr      nz,acesetor1
                bit     4,a             ;houve erro de busca?
                jr      nz,comerro      ;Sim, vai para comerro
                bit     3,a             ;houve erro de CRC?
                jr      nz,comerro      ;Sim, vai para comerro
                                        ;Não, então houve perda de
                                        ;dados
comerro
                ld      a,#01


;a rotina prepfim prepara a saída de um processo
;de leitura ou gravação de setores

prepfim
                push    af              ;salva a
                ld      a,#d0           ;força o término
                out     (#d0),a         ;das operações
                                        ;anteriores
                call    testcond        ;espera liberação
                pop     af              ;recupera a
                ret
```

```
;rotina para leitura de um setor. hl=núm. do setor, de=end.
;em RAM do buffer para o setor

lesetor
            ld      ix,parleitura       ;ix aponta para os parâmetros
                                        ;de leitura
            ld      a,(ix+#00)          ;a=comando de leitura
            ld      (acesetor2+1),a     ;modifica a rotina acesetor
            ld      a,(ix+#01)          ;a=instrução ini (leitura)
            ld      (acesetor5+1),a     ;modifica a rotina acesetor
            ld      a,(ix+#02)          ;a=parâmetro de verificação
                                        ;da leitura
            ld      (acesetor6+1),a     ;modifica a rotina acesetor
lesetor_0   ld      ix,pardrivea        ;ix -> tabela drive a
            ld      a,(drive)           ;
            dec     a                   ;drive a?
            jr      z,lesetor_1         ;Sim, pula para lesetor_1
            ld      ix,pardriveb        ;Não, atualiza ix
lesetor_1
            push    hl                  ;salva hl
            push    bc                  ;salva bc
            push    de                  ;salva de
            di                          ;
            call    inidrive            ;faz a leitura de um setor
            ei                          ;
            pop     de                  ;recupera o buffer
            ld      hl,(numbytes)       ;soma com o número de bytes
            add     hl,de               ;por setor
            ex      de,hl               ;
            pop     bc                  ;recupera os outros registros
            pop     hl                  ;
            inc     hl                  ;aponta para o próximo setor
            djnz    lesetor_1           ;lê os outros setores
            or      a                   ;houve erro?
            ret     z                   ;Não, volta.
            ld      hl,#ffff            ;Sim, sinaliza ao BASIC
            ld      (argusr),hl         ;
            ret                         ;e retorna

parleitura
            defb    #80,#a2,#9c


;rotina para a gravação de um setor. hl=núm. do setor,
;de=end. em RAM do buffer do setor

gravsetor
```

```
            ld      ix,parescrita       ;ix aponta para os parâmetros
                                        ;de escrita
            ld      a,(ix+#00)          ;a=comando de escrita
            ld      (acesetor2+1),a     ;modifica a rotina acesetor
            ld      a,(ix+#01)          ;a=instrução outi (escrita)
            ld      (acesetor5+1),a     ;modifica a rotina acesetor
            ld      a,(ix+#02)          ;a=parâmetro de verificação
            ld      (acesetor6+1)       ,a;da escrita
gravset_0   ld      ix,pardrivea        ;ix -> tabela drive a
            ld      a,(drive)           ;
            dec     a                   ;drive a?
            jr      z,gravset_1         ;Sim, pula para gravset_1
            ld      ix,pardriveb        ;Não, atualiza ix
gravset_1
            push    hl                  ;salva os registros
            push    bc                  ;
            push    de                  ;
            di                          ;
            call    inidrive            ;grava um setor
            ei                          ;
            pop     de                  ;recupera o buffer
            ld      hl,(numbytes)       ;soma com o número de bytes
            add     hl,de               ;por setor
            ex      de,hl               ;
            pop     bc                  ;recupera os outros registros
            pop     hl                  ;
            inc     hl                  ;aponta para o setor seguinte
            djnz    gravset_1           ;grava os outros setores
            or      a                   ;houve erro?
            ret     z                   ;Não, volta
            ld      hl,#ffff            ;Sim, sinaliza ao BASIC
            ld      (argusr),hl         ;
            ret                         ;e retorna

parescrita
            defb    #a0,#a3,#fc


pardrivea
            defb    #21                 ;parâmetro p/ ligar
            defb    #00                 ;trilha atual
            defb    #01                 ;densidade
            defb    #f9                 ;tipo de formatação
```

```
pardriveb
                defb    #22             ;parâmetro p/ ligar
                defb    #00             ;trilha atual
                defb    #01             ;densidade
                defb    #f9             ;tipo de formatação


;rotina que pára o motor do drive sem mexer no cabeçote

paradrive
                push    af              ;salva o registro afetado
                xor     a               ;zera o registro a
                out     (#d4),a         ;envia o comando de parada
                pop     af              ;recupera o registro
                ret                     ;retorna


;rotina para deslocar o cabeçote até a trilha zero e
;parar o drive. Esta rotina deve ser usada sempre que
;se desejar parar o drive

rodadrive
                di                      ;desabilita as interrupções
                push    af              ;salva os registros afetados
                push    bc
                push    de
                push    hl
                ld      a,(drive)       ;obtém o número do drive
                ld      e,#20           ;comando para ligar o motor
                or      a               ;a=comando para acionar
                out     (#d4),a
                call    testcond        ;espera liberação
                ld      a,#02           ;a=comando de inicialização
                out     (#d0),a         ;envia o comando
                call    testcond        ;espera liberação
                ld      b,200           ;dá um tempo para o ajuste
                call    espera          ;das partes mecânicas
                call    paradrive       ;pára o motor
                pop     hl              ;recupera os registros
                pop     de
                pop     bc
                pop     af
                ei                      ;habilita as interrupções
                ret                     ;retorna
```

```
;rotina para estabilização das partes mecânicas do drive

espera
                ex      (sp),hl
                ex      (sp),hl
                djnz    espera
                ret


;rotina para testar a liberação do FDC

testcond
                ex      (sp),hl
                ex      (sp),hl
                ex      (sp),hl
                ex      (sp),hl
testcond_1
                in      a,(#d0)
                rrca
                jr      c,testcond_1
                ret


;área das variáveis usadas no código-fonte

numbytes        defw    #0000
numsetor        defb    #00
numset          defb    #00
drive           defb    #00
confatual       defb    #00
subsatual       defb    #00
slotatual       defb    #00
vez             defb    #00

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa para cópia de setores:

```
10 FOR A%=&HD500 TO &HD7FE
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
```

```
50 BSAVE "PROG33.BIN",&HD500,&HD7FE
100 DATA F3,CD,38,01,32,FA,D7,AF,21,C1,FC,06,04,32,FC,D7
110 DATA 32,FB,D7,CB,7E,20,27,CD,64,D5,38,0C,23,3A,FC,D7
120 DATA 3C,10,EA,21,FF,FF,18,06,3A,FB,D7,6F,26,00,3E,02
130 DATA 32,63,F6,22,F8,F7,3A,FA,D7,CD,3B,01,FB,C9,C5,E5
140 DATA 1E,00,06,04,7B,17,17,E6,0C,5F,3A,FC,D7,E6,03,B3
150 DATA CB,FF,32,FB,D7,CD,64,D5,38,03,1C,10,E7,E1,C1,D2
160 DATA 1C,D5,18,C4,C5,D5,E5,CD,99,D5,F5,D3,8E,1E,00,CD
170 DATA A3,D5,DB,8E,1E,EB,CD,A3,D5,D3,8E,1E,00,CD,A3,D5
180 DATA CD,99,D5,FE,EB,28,08,F1,5F,CD,A3,D5,AF,18,06,F1
190 DATA 5F,CD,A3,D5,37,E1,D1,C1,C9,21,00,40,3A,FB,D7,CD
200 DATA 0C,00,C9,21,00,40,3A,FB,D7,CD,14,00,C9,F3,CD,E1
210 DATA D5,DB,8E,21,00,94,11,00,40,01,00,40,ED,B0,D3,8E
220 DATA 3A,FA,D7,CD,3B,01,FB,C9,F3,CD,E1,D5,21,00,40,11
230 DATA 00,94,01,00,40,ED,B0,D3,8E,3A,FA,D7,CD,3B,01,FB
240 DATA C9,CD,38,01,32,FA,D7,3A,FB,D7,21,00,40,CD,24,00
250 DATA D3,8E,3A,F8,F7,32,00,40,3C,32,00,60,C9,3A,FD,D7
260 DATA B7,20,0B,21,00,02,22,F5,D7,3E,09,32,F7,D7,3A,F8
270 DATA F7,32,F9,D7,CD,C1,D7,ED,5B,51,F3,21,00,00,06,01
280 DATA CD,2D,D7,CD,BB,D7,3A,FD,D7,B7,C0,DD,2A,51,F3,DD
290 DATA 6E,0B,DD,66,0C,22,F5,D7,DD,7E,18,32,F7,D7,DD,7E
300 DATA 15,F5,DD,7E,1A,F5,DD,21,B3,D7,FD,21,B7,D7,F1,DD
310 DATA 77,02,FD,77,02,F1,DD,77,03,FD,77,03,DD,36,01,00
320 DATA FD,36,01,00,3E,01,32,FD,D7,C9,11,00,94,2A,F8,F7
330 DATA 3A,F8,D7,47,CD,2D,D7,C9,11,00,94,2A,F8,F7,3A,F8
340 DATA D7,47,CD,70,D7,C9,D5,CD,EB,D7,3A,F7,D7,4F,06,08
350 DATA 29,7C,91,38,02,67,2C,10,F7,24,DD,7E,00,5F,DD,7E
360 DATA 02,3D,7B,28,06,CB,3D,30,02,F6,10,57,DD,7E,03,0F
370 DATA 0F,E6,C0,B2,7A,D3,D4,CD,EB,D7,7C,D3,D2,CD,EB,D7
380 DATA DD,7E,01,D3,D1,BD,28,12,7D,D3,D3,DD,77,01,3E,11
390 DATA D3,D0,CD,EB,D7,06,10,CD,E6,D7,E1,1E,05,E5,D5,CD
400 DATA EB,D7,3E,80,CB,72,28,08,F6,02,CB,62,28,02,F6,08
410 DATA D3,D0,06,03,CD,E6,D7,0E,D3,11,0A,D7,D5,DB,D0,0F
420 DATA D0,0F,D2,FD,D6,ED,A2,C3,FD,D6,D1,E1,DB,D0,E6,9C
430 DATA 28,11,CB,77,20,0B,1D,20,C4,CB,67,20,04,CB,5F,20
440 DATA 00,3E,01,F5,3E,D0,D3,D0,CD,EB,D7,F1,C9,DD,21,6D
450 DATA D7,DD,7E,00,32,E3,D6,DD,7E,01,32,06,D7,DD,7E,02
460 DATA 32,0F,D7,DD,21,B3,D7,3A,F9,D7,3D,28,04,DD,21,B7
470 DATA D7,E5,C5,D5,F3,CD,86,D6,FB,D1,2A,F5,D7,19,EB,C1
480 DATA E1,23,10,ED,B7,C8,21,FF,FF,22,F8,F7,C9,80,A2,9C
490 DATA DD,21,B0,D7,DD,7E,00,32,E3,D6,DD,7E,01,32,06,D7
500 DATA DD,7E,02,32,0F,D7,DD,21,B3,D7,3A,F9,D7,3D,28,04
510 DATA DD,21,B7,D7,E5,C5,D5,F3,CD,86,D6,FB,D1,2A,F5,D7
520 DATA 19,EB,C1,E1,23,10,ED,B7,C8,21,FF,FF,22,F8,F7,C9
530 DATA A0,A3,FC,21,00,01,F9,22,00,01,F9,F5,AF,D3,D4,F1
540 DATA C9,F3,F5,C5,D5,E5,3A,F9,D7,1E,20,B7,D3,D4,CD,EB
550 DATA D7,3E,02,D3,D0,CD,EB,D7,06,C8,CD,E6,D7,CD,BB,D7
```

```
560 DATA E1,D1,C1,F1,FB,C9,E3,E3,10,FC,C9,E3,E3,E3,E3,DB
570 DATA D0,0F,38,FB,C9,00,00,00,00,00,00,00,00,00,00
```

Listagem do programa de teste:

```
1000 KEYOFF:WIDTH40:CLEAR 200,&H93FF:DEFINTA-Z
1010 CLS:BLOAD"PROG25.BIN",R:BLOAD"PROG33.BIN"
1020 DEFUSR0=&HD500:DEFUSR1=&HD5AD:DEFUSR2=&HD5C8
1030 DEFUSR3=&HD5FD:DEFUSR4=&HD66A:DEFUSR5=&HD678
1040 DEFUSR6=&HD7C1:DEFUSR7=&HD7BB:DR=&HD7F9:NT=&HD7F8:VZ=
&HD7FD
1050 GOSUB1780
1060 IFA=1THENENDELSECLS
1070 ED!=PEEK(&HF351)+256*PEEK(&HF352)
1080 DEFFNN(ED!)=(PEEK(ED!+19)+256*PEEK(ED!+20))
1090 A=USR0(0):IFA=-1THENGOTO2110
1100 GOSUB1130
1110 IFDD=DFTHENGOSUB1200ELSEGOSUB1340
1120 GOTO 1050
1130 S1$="DRIVE-FONTE":S2$="DRIVE -CÓPIA"
1140 S3$="ESCOLHA OS DRIVES":X1=(40-LEN(S3$))/2-1:X2=X1+1+
LEN(S3$):Y1=9:Y2=11
1150 WINDOWX1,Y1,X2,Y2,1:LOCATEX1+1,Y1+1:PRINTS3$:LOCATE1,0
:PRINTS1$:LOCATE21,0:PRINTS2$:C=0:D=0
1160 MENU1,0,21,0,11,9,A
1170 IFA=0THENGOSUB1480:DF=A+1:C=1:IFD=1THENRETURNELSEGOTO
1160
1180 IFA=1THENGOSUB1480:DD=A+1:D=1:IFC=1THENRETURN
1190 GOTO1160
1200 DA=DF:GOSUB1960:CLS:POKEVZ,0
1210 A=USR3(DF):N1=FNN(ED!)
1220 X1=0:X2=24:Y1=2:Y2=8
1230 WINDOWX1,Y1,X2,Y2,1
1240 LOCATEX1+1,Y1+2:PRINT"DISCO-FONTE : ";CHR$(&H40+DF):
LOCATEX1+1,Y2-2:PRINT"NÚM. DE SETORES:";N1
1250 DA=DD:GOSUB1980
1260 A=USR3(DD):N2=FNN(ED!)
1270 X1=12:X2=36:Y1=10:Y2=16
1280 WINDOWX1,Y1,X2,Y2,1
1290 LOCATEX1+1,Y1+2:PRINT"DISCO DA CÓPIA : ";CHR$(&H40+DD)
:LOCATEX1+1,Y2-2:PRINT"NÚM. DE SETORES:";N2
1300 GOSUB2080
1310 IFN1<>N2THENGOTO1840
1320 GOSUB1590
1330 RETURN
1340 POKEVZ,0:S1$="COLOQUE OS DISQUETES NOS DRIVES"
```

```
1350 GOSUB1900:CLS
1360 A=USR3(DF):N1=FNN(ED!):A=USR3(DD):N2=FNN(ED!)
1370 X1=0:X2=24:Y1=2:Y2=8
1380 WINDOWX1,Y1,X2,Y2,1
1390 LOCATEX1+1,Y1+2:PRINT"DRIVE : ";CHR$(&H40+DF)
1400 LOCATEX1+1,Y2-2:PRINT"NÚM. DE SETORES:";N1
1410 X1=X1+12:X2=X2+12:Y1=Y1+8:Y2=Y2+8
1420 WINDOWX1,Y1,X2,Y2,1
1430 LOCATEX1+1,Y1+2:PRINT"DRIVE : ";CHR$(&H40+DD):
LOCATEX1+1,Y2-2:PRINT"NÚM. DE SETORES:";N2
1440 GOSUB2080
1450 IFN1<>N2THENGOTO1840
1460 GOSUB1590
1470 RETURN
1480 IFA=0THENX1=1ELSEX1=21
1490 X2=X1+8:Y1=1:Y2=4
1500 LOCATEX1-1,Y1+1:PRINT" ":LOCATEX1-1,Y1+2:PRINT" ":
LOCATEX2+1,Y1+1:PRINT" ":LOCATEX2+1,Y1+2:PRINT" "
1510 REVERSE,,0,1
1520 WINDOWX1,Y1,X2,Y2,1
1530 LOCATEX1+1,Y1+1:PRINT"DRIVE A"
1540 LOCATEX1+1,Y1+2:PRINT"DRIVE B"
1550 MENUX1+1,Y1+1,X1+1,Y1+2,7,1,A
1560 REVERSE,,0,1
1570 LOCATEX1-1,Y1+1+A:PRINT">":LOCATEX2+1,Y1+1+A:PRINT"<"
1580 RETURN
1590 Q1=N1-(N1MOD512):Q2=N1-((N1-Q1)MOD32):Q3=N1
1600 FORX=0TOQ1-1STEP512
1610 DA=DF:PG=0:NS=32:POKEDR,DA:POKENT,NS:IFQ1=0THENQL=Q2-1EL
SEQL=X+511
1620 IFDF=DDTHENGOSUB1960
1630 FORY=XTOQLSTEP32:SI=Y:GOSUB2000:A=USR4(Y):IFA=-1THEN
2090ELSEA=USR1(PG):PG=PG+2:NEXTY
1640 IFQ1=0THENNS=N1-Q2:POKENT,NS:SI=Q2:GOSUB2000:A=USR4
(Q2):IFA=-1THEN2090ELSEA=USR1(PG)
1650 DA=DD:PG=0:NS=32:POKEDR,DA:POKENT,NS:IFDF=DDTHENGOSUB
1980
1660 FORY=XTOQLSTEP32:SI=Y:GOSUB2020:A=USR2(PG):A=USR5(Y):IFA=-
1THEN2100ELSEPG=PG+2:NEXTY
1670 IFQ1=0THENNS=N1-Q2:POKENT,NS:SI=Q2:GOSUB2020:A=USR2
(PG):A=USR5(Q2):IFA=-1THEN2100ELSEGOTO1770
1680 NEXTX
1690 DA=DF:PG=0:NS=32:POKEDR,DA:POKENT,NS:IFDF=DDTHENGOSUB1960
1700 FORY=Q1TOQ2-1STEP32:SI=Y:GOSUB2000:A=USR4(Y):IFA=-1
THEN2090ELSEA=USR1(PG):PG=PG+2:NEXTY
1710 IF(Q2=Q3ANDDF=DD)THENGOSUB1980
1720 IFQ2=Q3THENPG=0:DA=DD:POKEDR,DA:FORY=Q1TOQ2-1STEP32
```

```
:SI=Y:GOSUB2020:A=USR2(PG):A=USR5(Y):IFA=-1THEN2100ELSE
PG=PG+2:NEXTY:GOTO1770
1730 NS=Q3-Q2:POKENT,NS:SI=Q2:GOSUB2000:A=USR4(Q2):IFA=-1THEN2090EL
SEA=USR1(PG)
1740 DA=DD:PG=0:NS=32:POKEDR,DA:POKENT,NS:IFDF=DDTHENGOSUB
1980
1750 FORY=Q1TOQ2-1STEP32:SI=Y:GOSUB2020:A=USR2(PG):A=USR5
(Y):IFA=-1THEN2100ELSEPG=PG+2:NEXTY
1760 NS=Q3-Q2:POKENT,NS:SI=Q2:GOSUB2020:A=USR2(PG):A=USR5
(Q2)
1770 POKEDR,DF:A=USR6(0):POKEDR,DD:A=USR6(0):RETURN
1780 CLS:S1$="CÓPIA DE DISQUETES? SIM":S2$="NÃO"
1790 X1=(40-LEN(S1$))/2:Y1=11:LOCATEX1,Y1:PRINTS1$
1800 X1=X1+LEN(S1$)-3:LOCATEX1,Y1+1:PRINTS2$
1810 MENUX1,Y1,X1,Y1+1,3,1,A
1820 REVERSE,,0,1
1830 RETURN
1840 S1$="O DISQUETE DE CÓPIA É DIFERENTE":S2$="DO DISQUETE-FONTE"
1850 CLS:BEEP:X1=(40-LEN(S1$))/2:X2=X1+1+LEN(S1$):Y1=10:Y2=13
1860 WINDOWX1,Y1,X2,Y2,1
1870 LOCATEX1+1,Y1+1:PRINTS1$:LOCATE(40-LEN(S2$))/2,Y1+2
:PRINTS2$
1880 GOSUB2080:GOSUB1780
1890 IFA=1THENCLS:END:ELSECLS:RETURN1070
1900 S2$="PRESSIONE QUALQUER TECLA"
1910 X1=(40-LEN(S1$))/2-1:X2=X1+1+LEN(S1$):Y1=20:Y2=23
1920 GOSUB2070
1930 WINDOWX1,Y1,X2,Y2,1
1940 LOCATEX1+1,Y1+1:PRINTS1$:LOCATE(40-LEN(S2$))/2,Y2-1:
PRINTS2$
1950 IFINKEY$=""THEN1950ELSEGOSUB2070:RETURN
1960 S1$="COLOQUE O DISQUETE-FONTE EM "+CHR$(&H40+DA)+" "
1970 A=USR7(0):GOSUB1900:RETURN
1980 S1$="COLOQUE O DISQUETE DE CÓPIA EM "+CHR$(&H40+DA)+" "
1990 A=USR7(0):GOSUB1900:RETURN
2000 S1$="LENDO DO SETOR"+STR$(SI)+" AO"+STR$(SI+NS-1)
2010 GOSUB2040:RETURN
2020 S1$="GRAVANDO DO SETOR"+STR$(SI)+" AO"+STR$(SI+NS-1)
2030 GOSUB2040:RETURN
2040 CLS:X1=(40-LEN(S1$))/2-1:X2=X1+1+LEN(S1$):Y1=10:Y2=12
2050 WINDOWX1,Y1,X2,Y2,0
2060 LOCATEX1+1,Y1+1:PRINTS1$:RETURN
2070 FORA=Y1TOY2:LOCATE0,A:PRINTSTRING$(39," ");:NEXTA:
RETURN
2080 S1$="PRESSIONE QUALQUER TECLA":S2$="PARA CONTINUAR":
GOSUB1910:RETURN
2090 S1$="ERRO DE LEITURA":S2$="NO DRIVE "+CHR$(&H40+DF)
```

```
:A=USR6(DF):GOTO1850
2100 S1$="ERRO DE ESCRITA":S2$="NO DRIVE "+CHR$(&H40+DD)
:A=USR6(DD):GOTO1850
2110 S1$="SISTEMA SEM MEGARAM CONECTADA":S2$="IMPOSSÍVEL CONTI
NUAR":GOSUB1910:CLS:END
```

COMENTÁRIOS SOBRE O PROGRAMA PARA CÓPIA DE SETORES

Todo o controle do programa acima é feito pela listagem em BASIC apresentada como programa de teste. Mais uma vez, as minhas desculpas pela apresentação da listagem em BASIC de maneira tão confusa, mas não havia outro remédio, devido às limitações de memória já mencionadas no Capítulo 5. O programa acima está longe de ser o mais rápido dos copiadores, por apresentar um gerenciamento feito em BASIC. Se o programa fosse todo em linguagem de máquina, teríamos um ganho da ordem de 30%. Em todo caso, você poderá observar uma aceleração significativa neste processo de cópia em relação ao do programa 28. Gostaria apenas de ressaltar o uso dos pares IX e IY como ponteiros para tabelas com os parâmetros dos drives A e B, respectivamente. Estude com atenção o uso de tais ponteiros, já que o uso dele se torna imprescindível no acesso a 2 drives, devido à perda do registro #D1 do FDC quando se comuta de um drive para outro. Fora isso, as rotinas de acesso ao FDC são, basicamente, as mesmas empregadas no programa 32.

Para finalizar a nossa viagem pelo FDC, gostaria de apresentar a etapa de formatação de um disquete, já que se trata de uma etapa absolutamente necessária na utilização efetiva dele.

FORMATANDO DISQUETES

A formatação é a etapa que organiza o disquete para receber os dados. Esta organização é mais complicada do que pode parecer, tendo em vista as diferenças de velocidade de drive para drive. O ideal seria que todos eles *rodassem* a 300 rpm, mas este nem sempre é o caso, já que são admitidas variações de +/- 4,5 rpm. Como contornar essas diferenças? Para solucionar este problema, existe um seqüência de valores no início de cada trilha, que tem como função fazer o sincronismo da leitura com a velocidade em que os dados foram gravados. Este mecanismo se assemelha em muito ao da gravação do header, que antecede a gravação de um programa em fita cassete. Além disso, para que o sincronismo seja o mais perfeito possível, são incluídas faixas (gaps, em inglês) após os bytes de sincronismo. A partir deste momento, vamos chamar um conjunto de bytes de sincronismo de **bloco espaçador**. Justamente por causa desses gaps é que o processo de leitura de uma trilha resulta em erro. Após os gaps, temos um byte que identifica o início do endereço (ID). Logo após este indicador, temos o próprio registro de endereço (ID), que ocupa seis bytes. São eles:

Byte	Função
#00	Número da trilha
#01	Número do lado
#02	Número do setor
#03	Tipo de formato
#04 e #05	CRC (soma de controle)

Tabela 7.8: Estrutura do registro de endereço

O tipo de formato no registro de endereço é, na verdade, um multiplicador que informa o tamanho do setor. Vejamos alguns valores possíveis para o tipo de formato

Valor	Tamanho do setor
#00	128 bytes
#01	256 bytes
#02	512 bytes
#03	1024 bytes
#04	2048 bytes

Tabela 7.9: Valores para o tipo de formato no registro ID

Logo após o registro ID, temos outro bloco espaçador e outro gap. Após esses dois últimos blocos, temos um marcador assinalando o início do registro de dados (o setor). Após o setor, temos outra soma (CRC) de dois bytes. A Tabela 7.10 apresenta um mapa desta organização.

	Número de bytes (em hexa)	Valor em hexa	Função
	#3E	#4E	Bloco espaçador
	#0C	#00	Gap
	#03	#F6	
	#01	#FC	Marca de índice
	#32	#4E	Bloco espaçador
*	#0C	#00	Gap
*	#03	#F5	
*	#01	#FE	Marca de ID
*	#01	(#00 a #4F)	Número da trilha
*	#01	(#00 ou #01)	Lado
*	#01	(#01 a #09)	Número do setor
*	#01	#02	Tipo de formato
*	#01	#F7	(2 CRCs)
*	#16	#4E	Bloco espaçador
*	#0C	#00	Gap

Tabela 7.10: Mapeamento de trilha usado pelo MS-DOS e pelo MSX-DOS

	Número de bytes (em hexa)	Valor em hexa	Função
*	#03	#F5	
*	#01	#FB	Marca de end. de dados
*	#200	#40	Setor
*	#01	#F7	(2 CRCs)
*	#54	#4E	Bloco espaçador

*Onde as etapas assinaladas com * se repetem para os demais setores*

Tabela 7.10: Mapeamento de trilha usado pelo MS-DOS e pelo MSX-DOS(continuação)

Agora, um assunto que merece destaque. Você já viu que o campo ID fornece o número da trilha, o lado e o setor, mas qual será a melhor forma de disposição dos setores numa trilha? Será que a forma

1-2-3-4-5-6-7-8-9

é melhor do que a forma

1-3-5-7-9-2-4-6-8

no sentido de se obter uma maior velocidade de acesso? Vamos raciocinar juntos. Suponha que desejemos ler os setores 1 e 2 de uma determinada trilha. Por mais rápido que seja o Z 80, nunca conseguiremos que ele leia o setor 2 logo após o setor 1, com o disquete ainda na mesma volta (não se esqueça que o disquete está girando a 300rpm), se utilizarmos os métodos de acesso ao disco do MSX-DOS. Desta forma, para ler os setores 1 e 2 usando o primeiro formato, gastaríamos o tempo necessário para o disquete dar duas voltas. Já no segundo formato, como o setor 2 está bem à frente do setor 1, conseguiríamos ler os setores 1 e 2 com o disquete ainda na mesma volta. Pode parecer estranho, mas isto acelera a leitura e a gravação de arquivos em mais de 30% (não se esqueça que tudo termina em leitura e gravação de setores, aos olhos do FDC). A Tabela 7.11 fornece um quadro comparativo entre os dois formatos numa hipotética leitura dos 9 setores de uma trilha.

Número da volta	Setores lidos no formato 2	Setor lido no formato 1
1	1 e 2	1
2	3 e 4	2
3	5 e 6	3
4	7 e 8	4
5	9	5
6		6
7		7
8		8
9		9

Onde formato 1 :1-2-3-4-5-6-7-8-9
e formato 2 :1-3-5-7-9-2-4-6-8

Tabela 7.11: Número de voltas gasto na leitura de uma trilha

Com o auxílio do quadro acima, fica fácil perceber que a segunda forma de disposição dos setores numa trilha, é muito mais vantajosa, já que conseguimos ler toda uma trilha em 5 voltas, e não mais em 9. Será que você consegue inventar um método melhor? Tente a disposição 1-4-7-2-5-8-3-6-9! A disposição dos setores numa trilha recebe o nome de *interleave* e representa, como acabamos de ver, um fator ponderante na velocidade de acesso aos dados gravados nos discos. Devido à baixa velocidade do disk drive, o interleave médio é de 9:1, o que significa que, para se ler uma trilha num disquete formatado no padrão MS-DOS, gasta-se o tempo necessário para o disquete dar nove voltas (lembre-se que no padrão MS-DOS os setores estão dispostos seqüencialmente). Se adotarmos o formato 2 (1-3-5-7-9-2-4-6-8), o interleave baixa para 5:1, já que neste formato a leitura completa de uma trilha se faz em apenas 5 voltas. Desta forma, o desejável seria ter um interleave de 1:1, mas esta é uma façanha só alcançada em máquinas 286 e 386 (família IBM-PC), com clocks iguais ou acima de 20 MHz, equipadas com discos rígidos de 18 ms de tempo de acesso e placas controladoras com interleave específico de 1:1. É claro que num interleave de 1:1 os setores estão dispostos seqüencialmente (formato 1), mas as velocidades da placa controladora do disco rígido e do processador são tão grandes que dá para fazer uma leitura após a outra. No nosso MSX, sem previsão para disco rígido, temos de nos contentar com um interleave de 5:1, o que, convenhamos, já não é tão mau assim. Entretanto, devo observar que o formato 2 só leva vantagem sobre o formato 1 quando se utilizam os métodos do MSX-DOS (ou MS-DOS) para o acesso ao disco. Se utilizarmos o formato 2 e testarmos o disquete assim formatado com o programa 32 (formatação lógica), iremos observar que o processo de leitura dos setores se faz de maneira mais lenta do que num disquete que utiliza o formato 1 (disquete formatado pelo MSX-DOS). Qual o motivo de tal diferença? Acontece que, na leitura direta dos setores (como acontece no programa 32), a velocidade de processamento é muito alta, tendo em vista a ausência de cálculos demorados entre uma leitura e outra. Desta forma, embora o disquete esteja rodando a 300 rpm, o Z 80 é capaz de ler o próximo setor ainda na mesma volta. No caso do MSX-DOS, existe uma manipulação complicada da FAT entre uma leitura e outra de

um setor, o que por sua vez impede o Z 80 de conseguir ler o setor seguinte ainda na mesma volta, motivo pelo qual o formato 2 se torna mais vantajoso nesse caso.

Antes de passar às rotinas para formatação de uma trilha, gostaria de lhe transmitir um aviso: as listagens apresentadas abaixo formam um programa que formata uma determinada trilha, de acordo com a resposta do usuário. Desta forma, o disco formatado não será *bootável* se o usuário não formatar a trilha 0 e pedir explicitamente a criação de um boot. Devo também esclarecer que a formatação de apenas algumas trilhas é o método de proteção mais eficaz, já que inibe a cópia de setores devido aos inúmeros erros na tentativa de leitura de setores em trilhas que não estão sequer formatadas. Por outro lado, o disquete protegido deve possuir sempre a trilha 0 formatada, pois, caso contrário, será impossível dar o *boot* pelo disquete protegido. Agora, uma pergunta: de onde vamos tirar os dados e a rotina de partida a serem colocados no boot? Alguma sugestão? Bem, a ROM de 16 Kbytes na interface do drive deve possuir tais dados, pois, ao formatarmos um disquete, chamamos uma rotina nessa ROM, que, ao final da formatação, grava as informações necessárias no setor 0. "Em que endereço estão tais dados? Como acessá-los?" Os endereços são os seguintes:

Disquete	Tipo	Endereço dos dados do boot
5 1/4"	Face simples	#7BB6
5 1/4"	Face dupla	#7BD4
3 1/2"	Face simples	#7BF2
3 1/2"	Face dupla	#7C10

Tabela 7.12: Endereços dos dados do boot na ROM da interface controladora do disk drive

Você deve ter notado que a diferença entre um endereço e outro é #1E, que, como vimos no Capítulo 6, é exatamente o tamanho da área de dados no boot. Agora só resta saber o endereço da rotina de partida, não é?

Endereço da rotina de partida
na ROM da Interface do drive : **#7C45**

Bem, agora que já temos todas as informações necessárias, que tal implementar um formatador para disquetes de 5 1/4"? Melhor ainda: vamos implementar um formatador que apresente a opção de formatar determinadas trilhas apenas.

O PROGRAMA PARA FORMATAÇÃO DE DISQUETES DE 5 1/4"

O programa aqui apresentado permite a formatação de qualquer trilha, ou conjunto de trilhas, de um disquete de 5 1/4". Além desta característica, também é possível gravar ou não o boot. Antes, porém, gostaria de observar que, se você não gravar o boot, não poderá acessar o disquete em questão por intermédio do MSX-DOS (ou MS-DOS), tendo em vista que todas as características do disquete são gravadas no boot. Agora, outra observação: embora o programa esteja projetado para formatar somente disquetes de 5 1/4", as rotinas podem ser adaptadas (observe os comentários nas listagens) para a formatação de disquetes de 3 1/2". Se você quiser, poderá usar o programa, como apresentado, para formatar disquetes de 3 1/2" como se fossem de 5 1/4". Vamos às listagens.

Listagem em assembly Z-80 do código-fonte do programa para a formatação de disquetes de 5 1/4":

```
;programa para formatação de disquetes de 5 1/4"
;Compilar com o programa GEN80.COM usando a seguinte
;sintaxe:
;
;                GEN80 PROG34.BIN=PROG34.GEN
;
;onde PROG34.GEN é o nome do arquivo-texto com esta
;listagem

rdslt           equ     #000c
wrslt           equ     #0014
enaslt          equ     #0024
rslreg          equ     #0138
wslreg          equ     #013b
espelho         equ     #9400
slotdrv         equ     #f348
valtyp          equ     #f663
argusr          equ     #f7f8


                defb    #fe             ;simula em CP/M
                defw    inicio          ;o cabecalho de
                defw    fim             ;um arquivo
                defw    inicio          ;.BIN


                org     #d000

inicio
                ld      a,(valtyp)      ;o argumento é
```

```
            cp      #02             ;inteiro?
            ret     nz              ;Não, volta
            ld      a,(argusr)      ;arg. > 39 (mude para 80
                                    ;no caso de 3 1/2")
            cp      40              ;
            ret     nc              ;Sim, volta
inicio_1    ld      ix,pardrive     ;ajusta os ponteiros
            ld      iy,setores      ;para as duas tabelas
            ld      (ix+#00),#01    ;envia o setor inicial
            ld      (ix+#01),a      ;envia a trilha
            di                      ;desabilita as interrupções
            call    fazespelho      ;faz o espelho
            call    formatar        ;formata a trilha
            ei                      ;habilita as interrupções
            or      a               ;houve erro?
            ret     z               ;Não, volta
            ld      hl,#ffff        ;Sim, sinaliza
            ld      (argusr),hl     ;a ocorrência
            ret                     ;e volta


;rotina para posicionar o cabeçote do drive na trilha
;correta, preparando-o para a formatação de uma trilha

formatar
            call    prepfim         ;termina as operações
                                    ;anteriores
            ld      e,(ix+#02)      ;e=drive
            ld      a,(ix+#03)      ;a=lado
            sla     a               ;ajusta o lado
            sla     a               ;
            sla     a               ;
            sla     a               ;
            or      e               ;
            or      #20             ;parâmetro para
                                    ;ligar os motores
            out     (#d4),a         ;seleciona e liga
                                    ;o drive
            call    testcond        ;espera liberação
            ld      a,(ix+#00)      ;a=setor 1
            out     (#d2),a         ;informa o setor
            call    testcond        ;espera liberação
            ld      a,(ix+#01)      ;a=trilha
            out     (#d3),a         ;posiciona o
            ld      a,#11           ;cabeçote
            out     (#d0),a         ;
            call    testcond        ;espera liberação
```

```
                ld      b,#10           ;retardo para
                call    espera          ;estabilização
                ld      hl,espelho      ;
                ld      e,#03           ;e=número de
                                        ;tentativas
formata_0       push    hl              ;salva hl e de
                push    de              ;
                ld      de,fimform      ;endereço final
                push    de              ;
                ld      c,#d3           ;c=porta de dados
                ld      d,#4e           ;d=bloco espaçador
                ld      a,#f4           ;a=comando para
                                        ;gravação de uma
                out     (#d0),a         ;trilha
                ex      (sp),iy         ;retardo para
                ex      (sp),iy         ;sincronismo
formata_1       ld      a,(hl)          ;chegou ao
                inc     a               ;fim?
                jp      z,formata_3     ;Sim, vai para
                                        ;formata_1
formata_2       in      a,(#d0)         ;Não, espera
                rrca                    ;liberação
                ret     nc              ;Se acabou, sai
                rrca                    ;Se não, espera
                jp      nc,formata_2    ;liberação
                outi                    ;envia o byte
                jp      formata_1

formata_3
                in      a,(#d0)         ;terminou?
                rrca                    ;
                ret     nc              ;Sim, sai
                rrca                    ;Não, espera
                jp      nc,formata_3    ;liberação
                ld      a,d             ;envia espaçador
                out     (#d3),a         ;
                jp      formata_3       ;fecha o loop

fimform
                pop     de              ;recupera os
                pop     hl              ;registros
                in      a,(#d0)         ;testa o comando
                and     #7c             ;Tudo certo?
                jr      z,prepfim       ;Sim, termina
                bit     6,a             ;No caso de escrita, o disco
                jr      nz,comerro      ;estava protegido?
                dec     e               ;faz mais uma tentativa
```

```
                jr      nz,formata_0      ;
                bit     4,a               ;houve erro de busca?
                jr      nz,comerro        ;Sim, vai para comerro
                bit     3,a               ;houve erro de CRC?
                jr      nz,comerro        ;Sim, vai para comerro
                                          ;Não, então houve perda de
                                          ;dados
comerro
                ld      a,#01


;a rotina prepfim prepara a saída de um processo
;de leitura ou gravação

prepfim
                push    af                ;salva a
                ld      a,#d0             ;força o término
                out     (#d0),a           ;das operações
                                          ;anteriores
                call    testcond          ;espera liberação
                pop     af                ;recupera a
                ret


;a rotina fazespelho faz o espelho da trilha
;a ser formatada

fazespelho
                ld      hl,espelho        ;hl aponta para
                                          ;o buffer
                ld      b,#3e             ;faz o primeiro
                ld      a,#4e             ;bloco espaçador
                call    preenche_1        ;
                call    preenche          ;faz o gap
                ld      b,#03             ;
                ld      a,#f6             ;
                call    preenche_1        ;
                ld      (hl),#fc          ;
                inc     hl                ;
                ld      b,#32             ;faz o segundo
                ld      a,#4e             ;bloco espaçador
                call    preenche_1        ;
                ld      b,#09             ;b=9 setores
fazesp_1        push    bc                ;salva b
                call    preenche          ;faz o gap
                ld      b,#03             ;
```

```
              ld      a,#f5              ;
              call    preenche_1         ;
              ld      (hl),#fe           ;
              inc     hl                 ;
              ld      a,(ix+#01)         ;obtém a trilha
              ld      (hl),a             ;
              inc     hl                 ;
              ld      a,(ix+#03)         ;obtém o lado
              ld      (hl),a             ;
              inc     hl                 ;
              ld      a,(iy+#00)         ;obtém o setor
              ld      (hl),a             ;
              inc     hl                 ;
              inc     iy                 ;
              ld      (hl),#02           ;envia o tipo
              inc     hl                 ;
              ld      (hl),#f7           ;envia o CRC
              inc     hl                 ;
              ld      b,#16              ;envia outro
              ld      a,#4e              ;bloco espaçador
              call    preenche_1         ;
              call    preenche           ;
              ld      b,#03              ;
              ld      a,#f5              ;
              call    preenche_1         ;
              ld      (hl),#fb           ;
              inc     hl                 ;
              ld      bc,512             ;envia o setor
fazesp_2      ld      (hl),#40           ;
              inc     hl                 ;
              dec     bc                 ;
              ld      a,b                ;
              or      c                  ;
              jr      nz,fazesp_2        ;
              ld      (hl),#f7           ;
              inc     hl                 ;
              ld      b,#54              ;
              ld      a,#4e              ;
              call    preenche_1         ;
              pop     bc                 ;
              djnz    fazesp_1           ;
              ld      (hl),#ff           ;assinala o fim
              ret                        ;

preenche
              xor     a                  ;zera o registro a
              ld      b,#0c              ;b=tamanho do gap
```

```
preenche_1
                ld      (hl),a          ;
                inc     hl              ;
                djnz    preenche_1      ;
                ret


;a rotina fazboot grava na trilha 0 as informações necessárias
;para tornar o disquete bootável

fazboot
                call    rodadrive       ;desloca o cabeçote
                                        ;para a trilha 0
                xor     a               ;a aponta p/ trilha 0
                ld      (lado),a        ;lado 0
                call    inicio_1        ;formata a trilha 0
                ld      a,(tipo)        ;face simples?
                or      a               ;
                jr      z,fazboot_01    ;Sim, pula para fazboot_01
                ld      (lado),a        ;Não, atualiza o lado
                dec     a               ;e formata a trilha 0
                call    inicio_1        ;
fazboot_01
                call    rslreg          ;lê e salva a configuração
                push    af              ;atual dos slots
                ld      a,(slotdrv)     ;ativa a página 1 do
                ld      hl,#4000        ;slot onde se encontra
                call    enaslt          ;a interface do drive
                ld      hl,espelho      ;
                ld      de,espelho+#01  ;
                ld      b,#0c           ;b=núm. de setores privados
fazboot_0       push    bc              ;salva b
                ld      bc,#0200        ;preenche os setores
                ld      (hl),#00        ;privados com #00
                ldir                    ;
                pop     bc              ;
                djnz    fazboot_0       ;
                ld      hl,#7bb6        ;hl aponta para a área
                                        ;de dados (mudar para #fbf2
                                        ;no caso de 3 1/2")
                ld      a,(tipo)        ;obtém o tipo do disquete
                or      a               ;face simples?
                jr      z,fazboot_1     ;Sim, pula para fazboot_1
                ld      de,#001e        ;Não, ajusta hl para
                add     hl,de           ;a nova posição
fazboot_1       ld      de,espelho      ;transfere as informações
```

```
            ld      bc,#001e         ;
            ldir                     ;
            ld      hl,#7c45         ;transfere a rotina de
            ld      bc,#7d1b-#7c45   ;partida
            ldir                     ;
            pop     af               ;recupera a configuração
            call    wslreg           ;original dos slots
            ld      hl,espelho       ;
            ld      b,#02            ;b=número de FAT s
            ld      de,#0200         ;de=tamanho do setor
fazboot_2   push    bc               ;salva bc
            add     hl,de            ;hl aponta para o segundo
                                     ;setor
            push    hl               ;salva hl
            ld      a,(tipo)         ;obtém o tipo
            add     a,#fc            ;soma ao parâmetro (somar
                                     ;#f8 no caso de 3 1/2")
            ld      (hl),a           ;forma a FAT
            inc     hl               ;
            ld      (hl),#ff         ;
            inc     hl               ;
            ld      (hl),#ff         ;
            pop     hl               ;recupera hl
            add     hl,de            ;aponta para o terceiro
                                     ;setor (fazer nova soma
                                     ;"add hl,de" no caso de
                                     ;3 1/2"
            pop     bc               ;fecha o loop para formar
                                     ;a segunda FAT
            djnz    fazboot_2        ;
            ld      a,(tipo)         ;obtém o tipo do disquete
            ld      b,#09            ;b=número de setores
                                     ;privados em face simples
                                     ;trocar para #0b no caso
                                     ;de 3 1/2"
            or      a                ;face simples?
            jr      z,fazboot_3      ;Sim, vai para fazboot_3
            ld      b,#0c            ;b=número de setores
                                     ;privados em face dupla
                                     ;trocar para #0e no caso
                                     ;de 3 1/2"
fazboot_3
            ld      hl,#0000         ;hl=setor inicial
            call    grconj           ;grava os setores
                                     ;privados
            call    paradrive        ;pára o drive
            ret                      ;volta ao BASIC
```

```
;a rotina grconj grava um conjunto de até 32 setores
;(16 Kb) no disquete de destino

grconj
                ld      de,espelho      ;hl=end. inicial
                call    gravsetor       ;grava o(s) setor(es)
                ret                     ;e retorna


;rotina para posicionar o cabeçote do drive na trilha
;correta, preparando-o para a leitura ou escrita do
;setor especificado

inidrive
                push    de              ;salva o ptr. p/ buffer
                call    testcond        ;espera liberação
                ld      a,#09           ;a=núm. de setores por trilha
                ld      c,a             ;c=núm. de setores por trilha
                ld      b,#08           ;prepara a divisão em 8 bits
divide_1        add     hl,hl           ;do número do setor desejado
                ld      a,h             ;pelo número de setores
                sub     c               ;por trilha
                jr      c,divide_2
                ld      h,a
                inc     l
divide_2        djnz    divide_1
                inc     h               ;h=núm. do setor (1 a 9)
                ld      a,(ix+#00)      ;parâmetro para ligar
                ld      e,a             ;salva em e
                ld      a,(ix+#02)      ;obtém o núm. de lados
                dec     a               ;face simples?
                ld      a,e             ;
                jr      z,inidrive_1    ;Sim, vai para inidrive_1
                srl     l               ;divide a trilha por 2
                                        ;O número da trilha
                                        ;é ímpar (lado 1)?
                jr      nc,inidrive_1   ;Não, vai para inidrive_1
                or      #10             ;Sim, informa.


inidrive_1
                ld      d,a             ;d=seleção
                ld      a,(ix+#03)      ;a=tipo de form.
                rrca                    ;
                rrca                    ;
                and     #c0             ;
                or      d               ;nova seleção
```

```
            ld      a,d             ;salva em d
            out     (#d4),a         ;liga o motor do drive
            call    testcond        ;espera liberação
            ld      a,h             ;envia o setor
            out     (#d2),a         ;
            call    testcond        ;espera liberação
            ld      a,(ix+#01)      ;a=trilha anterior
            out     (#d1),a         ;envia o número
            cp      l               ;já está na nova trilha?
            jr      z,acesetor      ;Sim, vai para inidrive_2
            ld      a,l             ;Não, posiciona o
            out     (#d3),a         ;cabeçote na trilha
            ld      (ix+#01),a      ;atualiza a variável
            ld      a,#11           ;
            out     (#d0),a         ;
            call    testcond        ;espera liberação
            ld      b,#10           ;
            call    espera          ;


;a rotina acesetor serve aos propósitos de leitura e/ou
;gravação de um setor. Na entrada, de=buffer em RAM e
;hl=número do setor. Os labels drive, numsetor e densid
;devem estar corretamente preenchidos

acesetor
            pop     hl              ;recupera o ptr. p/ buffer
            ld      e,5             ;e=núm. de tentativas
acesetor1   push    hl              ;salva o end. do buffer
            push    de              ;salva núm. de tentativas
            call    testcond        ;espera liberação
acesetor2   ld      a,#80           ;a=comando de leitura
            bit     6,d             ;espera ativada?
            jr      z,acesetor3     ;Não, vai p/ acesetor3
            or      #02             ;habilita espera
            bit     4,d             ;é lado 1?
            jr      z,acesetor3     ;Não, vai para acesetor3
            or      #08             ;Sim, faz a comparação
                                    ;pelo lado 1
acesetor3   out     (#d0),a         ;envia o comando
            ld      b,#03           ;laço de espera para
            call    espera          ;sincronização
            ld      c,#d3           ;a=porta de dados do FDC
            ld      de,fimacesso    ;de=end. final
            push    de              ;salva end. final na pilha
acesetor4   in      a,(#d0)         ;lê status do FDC
```

```
                rrca                            ;acabou a leitura?
                ret         nc                  ;Sim, vai para fimacesso
                rrca                            ;Não, espera chegar um byte
                jp          nc,acesetor4
acesetor5       ini                             ;lê/escreve (outi) um byte
                jp          acesetor4           ;prepara nova leitura/escrita

fimacesso
                pop         de                  ;recupera núm. de tentativas
                pop         hl                  ;recupera end. do buffer
                in          a,(#d0)             ;obtém o status do FDC
acesetor6       and         #9c                 ;correu tudo bem?
                jp          z,prepfim           ;Sim, volta
                bit         6,a                 ;No caso de escrita, o disco
                jp          nz,comerro          ;estava protegido?
                dec         e                   ;faz mais uma tentativa
                jp          nz,acesetor1
                bit         4,a                 ;houve erro de busca?
                jp          nz,comerro          ;Sim, vai para comerro
                bit         3,a                 ;houve erro de CRC?
                jp          nz,comerro          ;Sim, vai para comerro
                                                ;Não, então houve perda de
                                                ;dados
                jp          comerro             ;


;rotina para a gravação de um setor. hl=núm. do setor,
;de=end. em RAM do buffer do setor

gravsetor
                ld          ix,parescrita       ;ix aponta para os parâmetros
                                                ;de escrita
                ld          a,(ix+#00)          ;a=comando de escrita
                ld          (acesetor2+1),a     ;modifica a rotina acesetor
                ld          a,(ix+#01)          ;a=instrução outi (escrita)
                ld          (acesetor5+1),a     ;modifica a rotina acesetor
                ld          a,(ix+#02)          ;a=parâmetro de verificação
                ld          (acesetor6+1)       ,a;da escrita
gravset_0       ld          ix,pardrivea        ;ix -> tabela drive a
                ld          a,(drive)           ;obtém o drive
                dec         a                   ;drive a?
                jr          z,gravset_1         ;Sim, pula para gravset_1
                ld          ix,pardriveb        ;Não, atualiza ix
gravset_1
                ld          a,(tipo)            ;obtém o tipo
                inc         a                   ;
```

```
                ld      (ix+#02),a      ;salva o lado
                add     a,#fb           ;
                ld      (ix+#03),a      ;salva o tipo
                push    hl              ;salva os registros
                push    bc              ;
                push    de              ;
                di                      ;
                call    inidrive        ;grava um setor
                ei                      ;
                pop     de              ;recupera o buffer
                ld      hl,#0200        ;soma com o número de bytes
                add     hl,de           ;por setor
                ex      de,hl           ;
                pop     bc              ;recupera os outros registros
                pop     hl              ;
                inc     hl              ;aponta para o próximo setor
                djnz    gravset_1       ;grava os outros setores
                or      a               ;houve erro?
                ret     z               ;Não, volta
                ld      hl,#ffff        ;Sim, sinaliza ao BASIC
                ld      (argusr),hl     ;
                ret                     ;e retorna

parescrita
                defb    #a0,#a3,#fc


;a rotina paradrive pára os motores de todos os
;drives

paradrive
                push    af              ;salva o registro a
                xor     a               ;pára todos os
                out     (#d4),a         ;drives
                pop     af              ;recupera o registro
                ret                     ;retorna


;rotina para estabilização das partes mecânicas do drive

espera
                ex      (sp),hl
                ex      (sp),hl
                djnz    espera
                ret
```

```
;rotina para testar a liberação do FDC

testcond
                ex      (sp),hl
                ex      (sp),hl
                ex      (sp),hl
                ex      (sp),hl
testcond_1
                in      a,(#d0)
                rrca
                jr      c,testcond_1
                ret


;rotina para deslocar o cabeçote até a trilha zero e
;parar o drive. Esta rotina deve ser usada sempre que
;se desejar parar o drive.

rodadrive
                di                          ;desabilita as interrupções
                push    af                  ;salva os registros afetados
                push    bc
                push    de
                push    hl
                ld      a,(drive)           ;obtém o número do drive
                ld      e,#20               ;comando para ligar o motor
                or      a                   ;a=comando para acionar
                out     (#d4),a
                call    testcond            ;espera liberação
                ld      a,#02               ;a=comando de inicialização
                out     (#d0),a             ;envia o comando
                call    testcond            ;espera liberação
                ld      b,200               ;dá um tempo para o ajuste
                call    espera              ;das partes mecânicas
                call    paradrive           ;pára o motor
                pop     hl                  ;recupera os registros
                pop     de
                pop     bc
                pop     af
                ei                          ;habilita as interrupções
                ret                         ;retorna


;área das variáveis usadas no código-fonte

pardrive
                defb    #00                 ;armazena o setor
```

```
                defb    #00                     ;armazena a trilha
drive           defb    #00                     ;armazena o número do drive
lado            defb    #00                     ;armazena o lado
tipo            defb    #00

setores
                defb    #01,#03,#05,#07,#09,#02,#04,#06,#08
pardrivea
                defb    #21                     ;parâmetro p/ ligar
                defb    #00                     ;trilha atual
                defb    #01                     ;densidade
                defb    #f9                     ;tipo de formatação


pardriveb
                defb    #22                     ;parâmetro p/ ligar
                defb    #00                     ;trilha atual
                defb    #01                     ;densidade
                defb    #f9                     ;tipo de formatação

fim             equ     $
```

Listagem em linhas DATA do código-objeto do programa para formatação de disquetes de 5 1/4":

```
10 FOR A%=&HD000 TO &HD304
20 READ B$
30 POKE A%,VAL("&H"+B$)
40 NEXT A%
50 BSAVE "PROG34.BIN",&HD000,&HD304
100 DATA 3A,63,F6,FE,02,C0,3A,F8,F7,FE,28,D0,DD,21,EE,D2
110 DATA FD,21,F3,D2,DD,36,00,01,DD,77,01,F3,CD,B8,D0,CD
120 DATA 2C,D0,FB,B7,C8,21,FF,FF,22,F8,F7,C9,CD,AE,D0,DD
130 DATA 5E,02,DD,7E,03,CB,27,CB,27,CB,27,CB,27,B3,F6,20
140 DATA D3,D4,CD,BF,D2,DD,7E,00,D3,D2,CD,BF,D2,DD,7E,01
150 DATA D3,D3,3E,11,D3,D0,CD,BF,D2,06,10,CD,BA,D2,21,00
160 DATA 94,1E,03,E5,D5,11,95,D0,D5,0E,D3,16,4E,3E,F4,D3
170 DATA D0,FD,E3,FD,E3,7E,3C,CA,87,D0,DB,D0,0F,D0,0F,D2
180 DATA 7A,D0,ED,A3,C3,75,D0,DB,D0,0F,D0,0F,D2,87,D0,7A
190 DATA D3,D3,C3,87,D0,D1,E1,DB,D0,E6,7C,28,11,CB,77,20
200 DATA 0B,1D,20,BF,CB,67,20,04,CB,5F,20,00,3E,01,F5,3E
210 DATA D0,D3,D0,CD,BF,D2,F1,C9,21,00,94,06,3E,3E,4E,CD
220 DATA 2F,D1,CD,2C,D1,06,03,3E,F6,CD,2F,D1,36,FC,23,06
230 DATA 32,3E,4E,CD,2F,D1,06,09,C5,CD,2C,D1,06,03,3E,F5
240 DATA CD,2F,D1,36,FE,23,DD,7E,01,77,23,DD,7E,03,77,23
```

```
250 DATA FD,7E,00,77,23,FD,23,36,02,23,36,F7,23,06,16,3E
260 DATA 4E,CD,2F,D1,CD,2C,D1,06,03,3E,F5,CD,2F,D1,36,FB
270 DATA 23,01,00,02,36,40,23,0B,78,B1,20,F8,36,F7,23,06
280 DATA 54,3E,4E,CD,2F,D1,C1,10,AF,36,FF,C9,AF,06,0C,77
290 DATA 23,10,FC,C9,CD,C9,D2,AF,32,F1,D2,CD,0C,D0,3A,F2
300 DATA D2,B7,28,07,32,F1,D2,3D,CD,0C,D0,CD,38,01,F5,3A
310 DATA 48,F3,21,00,40,CD,24,00,21,00,94,11,01,94,06,0C
320 DATA C5,01,00,02,36,00,ED,B0,C1,10,F5,21,B6,7B,3A,F2
330 DATA D2,B7,28,04,11,1E,00,19,11,00,94,01,1E,00,ED,B0
340 DATA 21,45,7C,01,D6,00,ED,B0,F1,CD,3B,01,21,00,94,06
350 DATA 02,11,00,02,C5,19,E5,3A,F2,D2,C6,FC,77,23,36,FF
360 DATA 23,36,FF,E1,19,C1,10,EC,3A,F2,D2,06,09,B7,28,02
370 DATA 06,0C,21,00,00,CD,BC,D1,CD,B4,D2,C9,11,00,94,CD
380 DATA 65,D2,C9,D5,CD,BF,D2,3E,09,4F,06,08,29,7C,91,38
390 DATA 02,67,2C,10,F7,24,DD,7E,00,5F,DD,7E,02,3D,7B,28
400 DATA 06,CB,3D,30,02,F6,10,57,DD,7E,03,0F,0F,E6,C0,B2
410 DATA 7A,D3,D4,CD,BF,D2,7C,D3,D2,CD,BF,D2,DD,7E,01,D3
420 DATA D1,BD,28,12,7D,D3,D3,DD,77,01,3E,11,D3,D0,CD,BF
430 DATA D2,06,10,CD,BA,D2,E1,1E,05,E5,D5,CD,BF,D2,3E,80
440 DATA CB,72,28,08,F6,02,CB,62,28,02,F6,08,D3,D0,06,03
450 DATA CD,BA,D2,0E,D3,11,46,D2,D5,DB,D0,0F,D0,0F,D2,39
460 DATA D2,ED,A2,C3,39,D2,D1,E1,DB,D0,E6,9C,CA,AE,D0,CB
470 DATA 77,C2,AC,D0,1D,C2,19,D2,CB,67,C2,AC,D0,CB,5F,C2
480 DATA AC,D0,C3,AC,D0,DD,21,B1,D2,DD,7E,00,32,1F,D2,DD
490 DATA 7E,01,32,42,D2,DD,7E,02,32,4B,D2,DD,21,FC,D2,3A
500 DATA F0,D2,3D,28,04,DD,21,00,D3,3A,F2,D2,3C,DD,77,02
510 DATA C6,FB,DD,77,03,E5,C5,D5,F3,CD,C3,D1,FB,D1,21,00
520 DATA 02,19,EB,C1,E1,23,10,E1,B7,C8,21,FF,FF,22,F8,F7
530 DATA C9,A0,A3,FC,F5,AF,D3,D4,F1,C9,E3,E3,10,FC,C9,E3
540 DATA E3,E3,E3,DB,D0,0F,38,FB,C9,F3,F5,C5,D5,E5,3A,F0
550 DATA D2,1E,20,B7,D3,D4,CD,BF,D2,3E,02,D3,D0,CD,BF,D2
560 DATA 06,C8,CD,BA,D2,CD,B4,D2,E1,D1,C1,F1,FB,C9,00,00
570 DATA 00,00,00,01,03,05,07,09,02,04,06,08,21,00,01,F9
580 DATA 22,00,01,F9,00
```

Listagem do programa de teste:

```
100 WIDTH 40:CLS:KEYOFF:CLEAR 200,&H93FF:DEFINTA-Z
110 BLOAD"PROG25.BIN",R:BLOAD"PROG34.BIN"
120 DEFUSR0=&HD000:DEFUSR1=&HD2B4:DEFUSR2=&HD134:DR=&HD2F0:
LD=&HD2F1:TP=&HD2F2
130 CLS:GOSUB 310:REM *** ESCOLHA DO DRIVE ***
140 A=A+1:POKE DR,A
150 CLS:GOSUB 340:REM *** ESCOLHA DO TIPO ***
160 T=A:POKE TP,A
```

```
170 CLS:GOSUB 370:REM *** ESCOLHA DAS TRILHAS ***
180 CLS:FOR TR=TI TO TF
190 POKE LD,0:REM *** INFORMA LADO 0 ***
200 GOSUB 420:REM *** INFORMA TRILHA E LADO ***
210 A=USR0(TR):REM *** FORMATA O LADO 0 DA TRILHA ***
220 IF A=-1 THEN GOTO 560:REM *** ASSINALA ERRO ***
230 IF T=0 THEN GOTO 280
240 POKE LD,1:REM *** INFORMA LADO 1 ***
250 GOSUB 420:REM *** INFORMA A TRILHA E O LADO ***
260 A=USR0(TR):REM *** FORMATA O LADO 1 DA TRILHA ***
270 IF A=-1 THEN GOTO 560:REM *** ASSINALA ERRO ***
280 NEXT TR
290 GOSUB 460:REM *** FIM DA FORMATAÇÃO ***
300 END
310 S1$="ESCOLHA O DRIVE":S2$="DRIVE A":S3$="DRIVE B"
320 GOSUB 490
330 RETURN
340 S1$="ESCOLHA O TIPO":S2$="FACE SIMPLES":S3$="FACE DUPLA"
350 GOSUB 490
360 RETURN
370 LOCATE 0,9:LINE INPUT"ENTRE COM A TRILHA INICIAL : ";A$
380 IF VAL(A$)>39 THEN 370 ELSE TI=VAL(A$)
390 LOCATE 0,11:LINE INPUT"ENTRE COM A TRILHA FINAL : ";A$
400 IF (VAL(A$)>39 OR VAL(A$)<TI) THEN 390
410 TF=VAL(A$):RETURN
420 S1$="FORMATANDO A TRILHA"+STR$(TR)+" LADO"+STR$(PEEK(
LD))
430 CLS:X1=(40-LEN(S1$))/2:Y1=11
440 LOCATE X1+1,Y1:PRINT S1$
450 RETURN
460 CLS:A=USR1(0):S1$="GRAVAR O BOOT":S2$="SIM":S3$="NÃO":
GOSUB 490
470 IF A THEN A=USR1(0) ELSE A=USR2(0)
480 CLS:S1$="FIM DA FORMATAÇÃO":GOSUB 430:END
490 X1=(40-LEN(S1$))/2:X2=X1+LEN(S1$):Y1=10:Y2=15
500 LOCATE X1,Y1-1:PRINT S1$
510 WINDOW X1-1,Y1,X2,Y2,1
520 X1=(40-LEN(S2$))/2
530 LOCATE X1,Y1+2:PRINT S2$:LOCATE X1,Y1+3:PRINT S3$
540 MENU X1,Y1+2,X1,Y1+3,LEN(S2$),1,A
550 RETURN
560 S1$="ERRO NA FORMATAÇÃO"
570 GOSUB 430:END
```

COMENTÁRIOS SOBRE O PROGRAMA PARA FORMATAÇÃO DE DISQUETES DE 5 1/4"

As listagens acima estão bem comentadas, de modo que você não encontrará maiores dificuldades para entendê-las. Apesar de tudo, gostaria de ressaltar que, além das modificações sugeridas na listagem do código-fonte, você deverá substituir o teste

IF VAL(A$)>39

nas linhas 380 e 400 da listagem do programa de teste, se quiser aplicar o programa em disquetes de 3 1/2". O teste deverá ser substituído por

IF VAL(A$)>79

já que os disquetes de 3 1/2" apresentam 80 trilhas, ao invés de 40.

Nunca é demais lembrar que todas as rotinas apresentadas neste capítulo foram testadas com sucesso em drives de 5 1/4" e de 3 1/2" controlados por interfaces do padrão MICROSOL (todas, menos a da SHARP).

NOVOS HORIZONTES

Apesar do MSX estar no mercado já há algum tempo, acho incrível como certas informações (como as apresentadas neste capítulo) ainda permanecem *secretas*. Creio, sinceramente, que um grande passo para o desenvolvimento da informática no Brasil reside na suspensão do tratamento infantil dado ao usuário nacional. Sinto que esta mentalidade está mudando, haja visto o nível crescente dos assuntos abordados nas revistas dedicadas aos microcomputadores MSX. Mas, mesmo assim, ainda existem artigos que simplesmente apresentam listagens enormes (o pior é que, na maioria das vezes, são listagens em hexadecimal) sem qualquer indicação dos algoritmos usados. Convenhamos que, assim, fica extremamente difícil para um usuário iniciante adquirir prática! Não se adquire prática, digitando listagens enormes do código-objeto; o máximo que se pode conseguir é que o usuário se transforme num ótimo digitador. Por outro lado, o enfoque dado ao MSX tem sido muito mais de um vídeo game do que de um ótimo computador, que é onde o MSX se encaixa.

Para os próximos anos, espero uma remodulação do parque do MSX, motivada pela migração do modelo 1.0 para o 2.0. Para tanto, já existem empresas nacionais fazendo a conversão (por sinal muito boa) do modelo 1.0 para o 2.0. Esta simples transformação abre um leque ainda maior de opções no desenvolvimento de programas mais amigáveis, baseados no novo chip de vídeo utilizado no MSX 2. Outro periférico que acabará se impondo no mercado, como aconteceu com o drive, é a MEGARAM. A partir do momento em que os produtores de software sentirem que o mercado deste periférico é grande, aparecerão verdadeiras maravilhas a nível de programas, antes só vistas em computadores de maior capacidade. Mas, para que tudo isto se torne realidade, é preciso haver uma maior publicação de informações, para dar subsídios ao surgimento de cada vez mais programadores, com novas idéias.

GUIA DO PROGRAMADOR MSX

O Que Este Livro Pretende.

Este livro pretende ser um guia completo para o programador do padrão MSX. Com este livro você será capaz de extrair o máximo do seu sistema. Entre os tópicos abordados, temos:

Acesso direto ao vídeo para um controle maior do processador de vídeo. As técnicas apresentadas neste tópico permitem a inversão de caracteres, criação de janelas, movimentos sobre áreas da tela, animação, etc.

Programação das interrupções para permitir uma simulação do mecanismo de multitarefa. Este tópico apresenta um programa muito interessante, que permite a visualização de uma parte da memória ao mesmo tempo em que se executa um outro programa.

Programação do vetor de erros para permitir a criação de novos comandos em BASIC. Este tópico apresenta listagens completas de programas que criam novos comandos para o BASIC, sem gastar um único byte da memória principal e sem fazer uso do comando DEFUSR para ativar os novos comandos.

Utilização do sistema de cartuchos para permitir uma utilização ampliada do interpretador BASIC em conjunto com as rotinas para desvio do vetor de erros.

Programação da MEGARAM. Este tópico é absolutamente inédito na literatura nacional. Aqui você aprenderá como acessar a memória de 256 Kbytes da sua MEGARAM usando rotinas e programas fartamente comentados.

Programação do Sistema MSX-DOS. Este tópico apresenta em detalhes todas as rotinas do sistema de disco MSX-DOS, acompanhadas de exemplos práticos de programação.

Acesso direto às portas do controlador de drives. Este tópico também é inédito na literatura nacional e apresenta todos os conhecimentos necessários para você programar diretamente os seus drives. Os programas de exemplo incluem um copiador e um formatador extremamente rápidos.

Sobre o Autor:

Eduardo Alberto Ramalho Sampaio Barbosa é engenheiro eletricista formado pela PUC/RJ, em Dezembro de 1986. Atualmente, exerce as funções de tradutor e programador free-lancer para as linhas de computadores IBM PC e MSX. É um entusiasta do MSX desde a chegada deste padrão ao Brasil, em fins de 1985. Sobre esta linha de computadores, já publicou dois livros: **Introdução à Linguagem de Máquina para MSX e Dicas, Macetes & Programas em Assembly para MSX Disk Drive,** ambos esgotados, publicados pela Editora Ciência Moderna.